BOLT BERANEK AND NEWMAN INC

CONSULTING • DEVELOPMENT • RESEARCH

BBN Report No. 2182
Job No. 11490

THE DEFENSE DOCUMENTATION CENTER

NATURAL ENGLISH PREPROCESSOR

W.A. Woods

31 July 1971

Submitted to:

Defense Supply Agency
Defense Documentation Center
Cameron Station
Alexandria, Virginia 22314

274

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Information Retrieval						
Bcolean Requests						
Natural Language Processing						
English						
Parsing						
Semantic Interpretation						

Unclassified

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Bolt Beranek and Newman Inc. 50 Moulton Street Cambridge, Massachusetts 02138	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP

3. REPORT TITLE

THE DEFENSE DOCUMENTATION CENTER NATURAL ENGLISH PREPROCESSOR

4. DESCRIPTIVE NOTES (Type of report and, inclusive dates)

Final Report (1 May 1970 - 30 July 1971)

5. AUTHOR(S) (First name, middle initial, last name)

W.A. Woods

6. REPORT DATE July 31, 1971	7a. TOTAL NO. OF PAGES 272	7b. NO. OF REFS 14
8a. CONTRACT OR GRANT NO. DAHC15-70-C-0233 b. PROJECT NO. c. d.	9a. ORIGINATOR'S REPORT NUMBER(S) BBN Report No. 2182 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)	

10. DISTRIBUTION STATEMENT

Distribution of this document is unlimited. It may be released to the Clearinghouse, Department of Commerce for sale to the general public.

11. SUPPLEMENTARY NOTES This research was sponsored by the Defense Documentation Center, Alexandria, Virginia 22314	12. SPONSORING MILITARY ACTIVITY

13. ABSTRACT

This report describes the results of a one-year project to develop a prototype system for translating natural English requests into a formal Boolean request language to give the Defense Documentation Center a look at the current state of the art in natural language processing as it would apply to the use of natural English queries in the DDC on-line environment. The system makes use of a transition network parser to perform a detailed syntactic analysis of the input sentences, and uses a general purpose semantic interpretation procedure to construct a Boolean combination of key phrases for use as a search strategy by the DDC retrieval system.

The performance of this system on two samples of 200 and 100 sentences respectively is described, and the major problems which must be solved before a practical production system could be implemented are discussed.

DD FORM 1473 (PAGE 1)

S/N 0101-807-6811

Unclassified

Security Classification

A-31409

THE DEFENSE DOCUMENTATION CENTER

NATURAL ENGLISH PREPROCESSOR

W.A. Woods

Bolt, Beranek, and Newman
Cambridge, Mass.

July 1971

CONTENTS

PREFACE

This report describes the results of a one-year project to develop a prototype system for translating natural English requests into a formal Boolean request language to give the Defense Documentation Center a look at natural language retrieval without involving extensive DDC personnel time and to establish a benchmark of accomplishment, which, if successful, would make it feasible for DDC to initiate an operational system.

Chapter 6 gives a concise summary of the project and its results with conclusions and recommendations for future development; Chapter 5 gives details of the performance analysis and evaluation and supports the conclusions of Chapter 6; Chapters 2 through 4 give a description of the components of the system, and their operation; and Chapter 1 gives an introduction to the project. Additional details are provided in the Appendices.

The casual reader will be primarily interested in Chapters 1 and 6, with additional details concerning the evaluation of performance in Chapter 5.

In addition to the author, the following staff have participated in the project:

Joe Becker

Dan Bobrow

Ron Kaplan

Bonnie Nash-Webber

Special acknowledgements are due to Ron Kaplan who wrote the majority of the grammar, and to Bonnie Nash-Webber who constructed

and debugged the dictionary and wrote many of the semantic interpretation rules.

Chapter 1

INTRODUCTION

1.1 Background and Objectives

The DDC English Language Preprocessor Prototype is an experimental prototype system for translating grammatically correct English language requests for information into Boolean combinations of words or phrases for retrieval from a descriptor oriented information retrieval data base. It takes input in the form of English sentences or noun phrases, and converts them into Boolean requests in a formal retrieval notation that approximates the Boolean request language of the DDC remote on-line retrieval system (DDC, 1970).

The prototype English preprocessor is intended to give DDC a feeling for the capabilities of natural language querying for information retrieval, and for the difficulties involved. From it one would hope to get a rough approximation of the performance that could be expected from an on-line interactive natural language querying system in which the original investigator (who has the need for the information) would ask his questions directly of the machine without the need for an intermediary. The system could then give him immediate response and allow him to rephrase his question in order to improve the quality of his retrieval.

The following paragraphs quoted from ANNEX "A" to the contract, specify the background and the objectives which motivated the implementation of the prototype English preprocessor. They provide the point of view from which this investigation of the feasibility of natural English querying was undertaken.

ANNEX "A"

CONTRACT NO. DAHC15 70 C 0233

PROTOTYPE NATURAL ENGLISH PREPROCESSOR
FOR BIBLIOGRAPHIC SEARCHING

I. BACKGROUND.

The Defense Documentation Center (DDC), Cameron Station, Alexandria, Virginia, is an agency of the Federal Government charged with being a central depository and distribution point for a wide variety of technical and scientific reports. The main DDC services center around the Technical Report Documentation System and the Management Information System. The volume of these services is already at a point where DDC makes extensive use of computers. In a bibliography request, for example, the user states his request on DDC Form 4 (Abstract Bibliography Request) and mails it to DDC, where the query is translated into the specialized query language of the DDC information retrieval (IR) system and processed by computer to effect the retrieval.

The increasing importance of timely response has led DDC to begin experimenting with an on-line IR system. Such a system implies that the user may interface directly with the DDC data base rather than to intermediaries who translate and format the query. To make such a system truly useful and responsive to users it must accept a query language that is easy and natural to use, rather than a formal type language dictated by the internal needs of the computer system. Computer acceptance of a wide range of natural English queries would result in a more streamlined batch system and thus in immediate improved service to present users.

DDC, therefore, sees a requirement for an investigation into the feasibility of natural English as a query language. Such a capability requires the development of a preprocessor capable of accepting natural English input and producing statements in the formal controlled information retrieval language of the present DDC System.

II. OBJECTIVES.

The goal of this research is to give DDC a realistic look at the suitability of natural English as an IR query language, without involving large numbers of DDC personnel in the design and implementation of a full-scale operational system. The use of natural language involves considerable linguistic talent, programmers, and programming skills to implement the technical linguistic features involved and considerable amounts of machine time for debugging, test, and evaluation. DDC realizes that the present state-of-the-art of linguistic processing makes it impossible to handle completely unrestricted English, but it hopes that the handling of English queries within a restricted field of discourse (specifically scientific prose) and limited to simple syntactic problems (simple and compound sentences and/or lists of keywords) is or can be made successful enough to present an economically viable approach to DDC's needs. The data generated by this research effort as to the technical feasibility and cost/effectiveness of natural English querying is expected to provide valuable data for the planning and development of the next generation DDC retrieval system, since it will cover at least half of the requests now received by DDC for its major data bank.

1.2 The Sample Query Set

A sample of 200 sentences was received from DDC as the initial data base for this project, with a target goal to parse and interpret 80% of this sample. The questions were drawn from the normal workload of requests which DDC receives by mail from its users. The sample contained a large diversity of syntactic constructions - larger even than one would have expected at first glance. Subtle differences in syntactic structure which are not noticeable to a human may be crucial to a rule driven parser, and we discovered a number of these in the course of developing the grammar.

The most apparent feature of the queries is that most of them are noun phrases rather than complete sentences. That is, there is an assumed prefix "GET me," "We need," etc. which is not stated. In those requests which are complete sentences, these prefixes are usually made explicit. In addition to the prefix "we need," there is another part of the request which may be missing or present depending on the requestor. This is the explicit reference to references or information on a topic.

The typical prototype request is of the form:

Give me We need etc.	a	bibliography list	of	information reports papers work etc.	on [specified topic]

As mentioned, most requestors omit the prefix "Give me," etc. Most requestors do explicitly state "information on" or the equivalent, and some explicitly use the words "list" or "bibliography." Some, however, omit the entire preamble and begin immediately with

a noun phrase which describes the specific topic. These phenomena are handled in the system by semantic rules which "understand" such constructions as "give me X", "information on Y", etc. and fill in appropriate defaults when these explicit constructions are missing.

Two major constructions in the sample, to which we have devoted considerable effort are the noun-noun modifiers (the use of a noun as an adjective to modify another noun) and reduced conjunctions (conjoined phrases in which common parts of the two conjuncts have been "factored out" to the left or right and appear only once). The former we have solved quite adequately, (e.g. "U.S. Naval vessel navigation equipment acquisition costs"). In the handling of reduced conjunctions, however, although we have achieved a facility which is significantly more advanced than anything else in the current state of the art of natural language processing, there are still ambiguities which seem unresolvable without extensive semantic information. Our current grammar is capable of parsing reduced conjunctions and enumerating all of the possible alternative parsings (of which there are usually a half-dozen or more) but we had hoped to be able to obtain the "most likely" parsing first, and thereby avoid having to enumerate the remaining alternatives. Indeed, our current strategy is to take the first parsing which "makes sense" (i.e. is interpretable to the semantic interpreter), and since the system lacks the semantic information to determine the sensibility of most topic titles, topic title parsings are all equally "sensible" and the interpreter in fact takes the first parsing which the parser enumerates. There are only a few cases in the sample where the system tries several parsings before finding one that is semantically interpretable.

The instances of conjunction in the sample are roughly equally distributed between totally unreduced conjunctions and conjunctions which are reduced to the maximum extent. Moreover, there does not seem to be any simple syntactic clue as to which of the two is the case. This problem is one for which we have yet found no satisfactory solution, and seems to require an extensive data base of semantic information to resolve. It is possible that some such information might be obtainable from the data base of acceptable DDC descriptors, but we have not had a chance to investigate this, nor would it have helped in the current prototype development since we have no direct connection to the DDC data base available.

Other characteristics of the samples include a large number of misspellings and many cases of rare and idiosyncratic grammatical errors on the part of the requestor (or some typist along the way). In many cases, the requestor has simply not said what he means, and in such cases the current system will interpret the sentence literally rather than in the way that the user intended. We do not consider this a serious problem in an on-line environment since the user can see when his request has been misinterpreted and rephrase it.

Broken down into syntactic types, the sample of 200 requests consists of 154 noun phrase utterances, 19 declarative sentences, 10 imperative sentences, 8 "pseudo imerative" sentences (the delted subject is "I"), 8 simple lists of keyphrases, and 1 question.

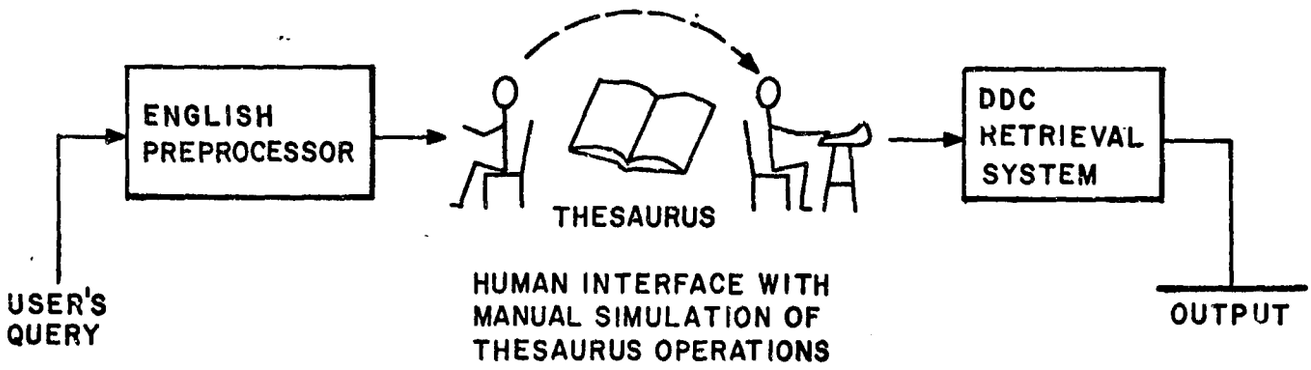
The English preprocessor system is viewed as ultimately operating in an environment in which the output of the semantic

interpreter would be fed directly to the DDC retrieval component, which would compare the request against a large thesaurus to standardize the usage of keywords and phrases, retrieve the appropriate references from its inverted files, and provide auxilliary information to the user concerning the number of items indexed under each of the keyphrases in the request. For the purpose of this initial experiment, however, the output from the semantic interpretation component is passed directly to a human observer for evaluation. In order to approximate the ultimate system performance, this human would have to manually perform the thesaurus operations and submit the result to the DDC on-line retrieval system for execution. These two modes of operation are pictured in figure 1-1.

1.4 Method of Approach

The method of approach which we have adopted in this study has been to look ahead to the potential capabilities for such a system, and to adopt general solutions to problems that will remain valid for applications of considerably greater scope than the mere conversion of English requests to Boolean combinations of phrases. We made this choice since it is clear that to take full advantage of the power and flexibility of natural language querying, a more sophisticated target language will be required. We have therefore chosen to embed the Boolean request generation facility within a general semantic framework (Woods, 1967, 1968) and to provide a comprehensive and rigorous grammar of the subset of English involved. We have used an existing parsing system for transition network grammars (Woods, 1969, 1970) to provide a powerful general parsing capability within reasonable amounts of processing time, and have operated on the resulting

a. ULTIMATE SYSTEM ORGANIZATION



b. EXPERIMENTAL APPROXIMATION

Figure 1-1. Ultimate and Experimental Environments for the DDC English Preprocessor

parse trees with a general rule driven semantic interpretation procedure (Woods, 1967, 1968) for transforming them into representations of their meanings. The only parts of the system which are specifically limited to the scope of the current task and would require major revisions for extending to more ambitious tasks are some of the specific semantic interpretation rules and the Boolean request generating functions that apply to the resulting semantic interpretations.

All of the components of the system have been implemented in BBN LISP on the PDP-10 computer at BBN in Cambridge, Mass., running under the TENEX time sharing system with hardware paging and a virtual core memory for each user of up to 256K. Although there is considerable overhead in running time for programs written in LISP and executed in a paged environment, the flexibility of this system has been a critical factor in the development of the present level of capability within the time scale of the contract. The techniques that we have used for natural language parsing and interpretation have consisted of "frontier" technology that is at the limits of the current state of the art or slightly beyond. In this type of development, it is not possible at the outset of a project to predict the consequences or the efficiency of a given processing technique on large samples of sentences (especially from an unconstrained environment) without extensive experimental testing of the system. As a result of such testing, therefore, one anticipates the necessity of making changes in the basic processing strategies in the course of developing such a system, and doing this in the most economical manner requires a programming language which permits such changes to be made easily without extensive effort devoted to providing basic facilities necessary to each change. LISP provides a basic repetory of symbol- and structure-manipulating facilities which permits such changes with effort devoted only to the essence of the changed strategy and not to large amounts of peripheral supporting facilities.

The design of the current system was carried out in a way that attempted to maximize the flexibility for such basic changes as: changing notations in dictionaries, changing parsing strategies, and modifying semantic interpretation rules and procedures; and indeed all of these have been changed extensively in the course of this project in order to achieve the current level of performance. Thus the current system represents the result of considerable evolution which would not have been possible within this time scale with a more rigid style of programming or a less flexible programming language.

Since the contract called for the implementation of the programs in a language that was operational on the UNIVAC 1108 at DDC, the programs were converted to 1108 LISP (Norman, 1969) at the end of the contract. No provision was made in the contract for testing or debugging these routines on the DDC system.

Chapter 2

THE ANALYSIS SYSTEM

2.1 Overview

In order to "understand" the intent of an English query, it is not sufficient to determine just the syntactic structure of the input sentence; rather it is necessary to determine the "meaning" of the sentence to the system, which is influenced both by the syntactic structure of the sentence and semantic information about the particular words which occur in it as they are related to the data base. In the prototype English preprocessor for the DDC retrieval system, this meaning is a Boolean combination of descriptors for use in retrieving documents. However, since a strictly Boolean descriptor language is not sufficiently rich to express the meanings of many queries which might naturally be directed toward the DDC data base, (e.g. what authors have more than three documents...) we have utilized a more powerful intermediate semantic language (similar to that described in Woods, 1968) to represent the semantic interpretations of the queries. We then map this representation into the simpler Boolean language whenever it can be so represented. This intermediate semantic representation allows for an analysis of the query that can be used to provide appropriate diagnostics in the case of queries which call for facilities not currently available in the retrieval system (such as sorting, and collating references) or for semantic errors (such as vacuous queries due to incompatible descriptors). The system processes Engligh queries in three successive phases:

(i) syntactic analysis using heuristic information to select the most "likely" parsings,

(ii) semantic interpretation to produce a formal representation of the "meaning" of the query to the system,

(iii) analysis of this representation and as appropriate translation to the formal Boolean request and/or error diagnostics.

The English Language preprocessor makes use of a general parsing algorithm for transition network grammars and a general rule-driven semanitc interpretation procedure which were developed at Harvard University and BBN over a period of years from 1967 to 1970, and which have been reported on in the literature (Woods, 1967, 1968, 1969, 1970). For this contract, we have adapted these programs to the DDC application, developed a grammar for a large subset of English (based on a sample of 200 sentences provided by DDC), developed a set of semantic interpretation rules for generating Boolean requests from parsed sentences, and constructed a large dictionary of approximately 4000 words. In addition, we have provided functions for converting from an arbitrary Boolean combination of keyphrases to a variation of "conjunctive normal form" which closely approximates the input syntax of the DDC retrieval system but is more inclusive (i.e. there are possible Boolean combinations of keyphrases which cannot be represented in the DDC request language but which might none-the-less be reasonable queries.) The overall organization of the English Language Preprocessor is shown in Figure 2-1. In this chapter we will be given a basic description of the operation of the major components of the system; a complete and detailed description of the individual functions which make up the system is given in Appendix D.

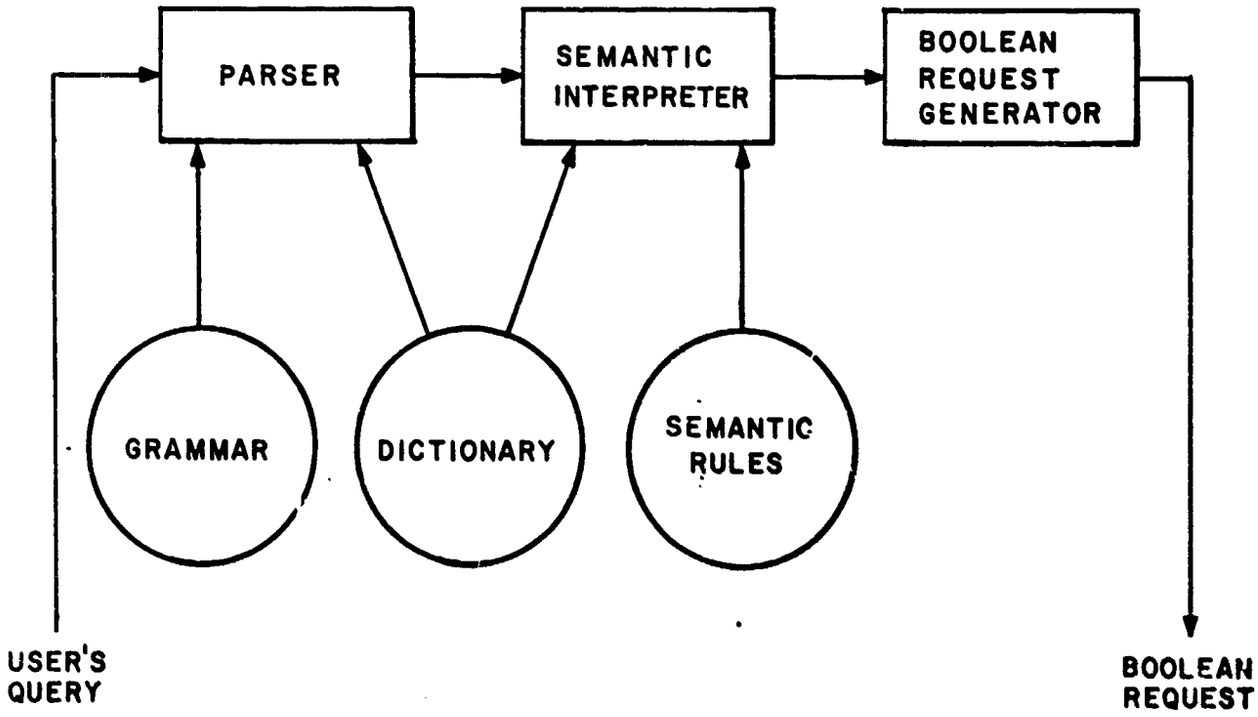


Figure 2-1. Organization of the English Language Preprocessor

2.2 The Parsing System

The DDC English preprocessor makes use of a general parsing procedure for transition network grammars developed at Harvard University and extended at Bolt, Beranek, and Newman (Woods, 1969 & 1970). This section gives a basic description of the transition network grammar model and the operation of the parsing system. (For more detail see Appendix D.) For more detail on the philosophy and motivation of the transition network grammar model, see the above cited references.

The transition network grammar model is an extension of the notion of state transition diagram well-known to automata theory. A transition network grammar consists of a network of nodes with arcs connecting them. The nodes represent states of a hypothetical parsing machine and the arcs connecting them represent possible transitions and are labelled with the types of events in the environment of the machine which permit the transitions. In the case of a transition network grammar, the types of events are the occurrences of words and phrases in the input string upon which the grammar is operating.

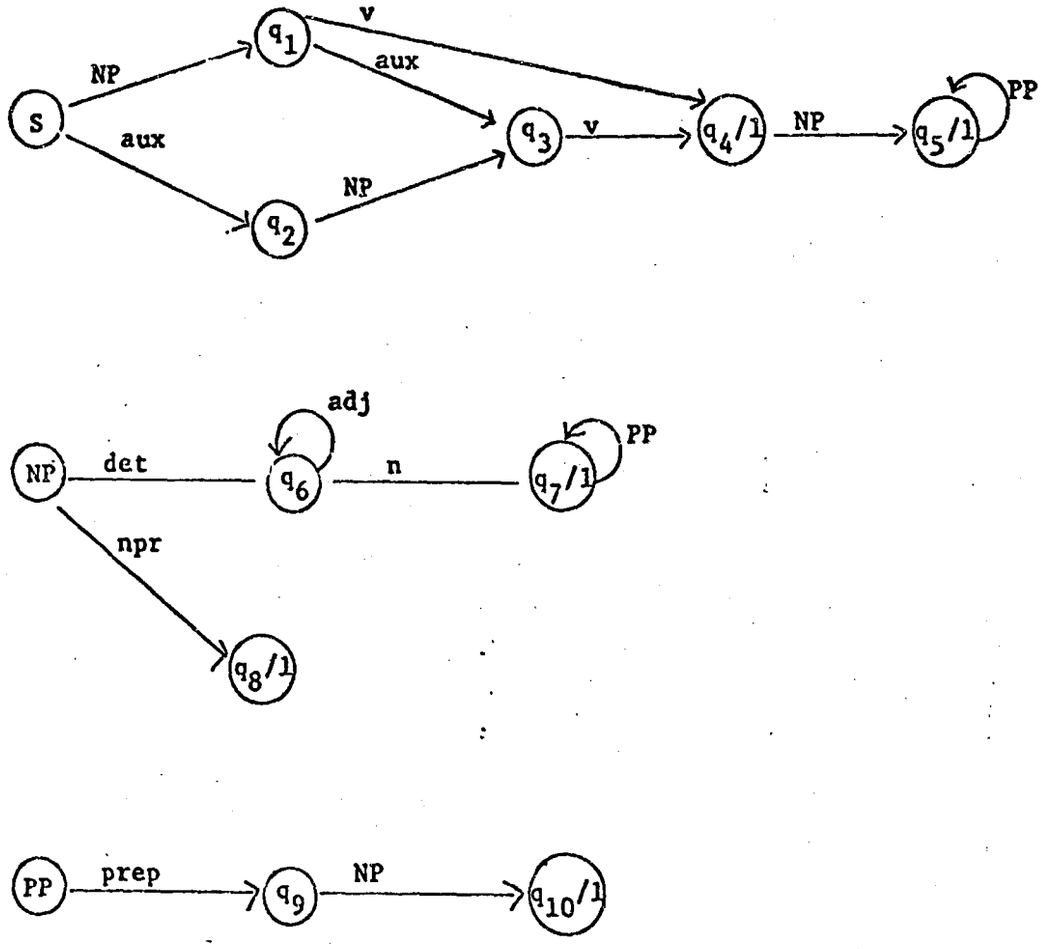
The type of transition network grammar which we are using for the DDC grammar is an augmented recursive transition network grammar in which the arcs of the network include arbitrary conditions for determining when their transitions are permitted, and arbitrary structure-building actions which build up the syntactic representation of the sentences recognized. This model has only recently been applied in the field of natural language processing, and it provides a practically feasible means of obtaining the types of analyses formerly obtainable only from laborious inversions of transformations specified by a transformational grammar of the

Chomsky variety (Chomsky, 1965). The transition network model permits analyses effectively equivalent to those of the transformational grammar, and it is the first parsing procedure to enable such sophisticated linguistic principles to be embodied in a practically feasible manner.

We say a state accepts a given string if that string permits a sequence of transitions which lead from that state to some state which is distinguished as a "final" state (in our system, this is indicated by the presence of a POP arc which not only marks the state as being a final state, but orders the alternative of accepting the string at that point with respect to the other arcs which leave that state).

A recursive transition network grammar contains two types of arcs--lexical arcs which correspond to transitions permitted by single words, and recursion arcs (or PUSH arcs) which invoke recursive applications of the network to recognize a phrase or word gouping of some kind. The most common type of the former is the CAT arc which recognizes members of a specified syntactic category. For example a CAT N arc permits a transition if the current word in the input string is a word in the syntactic category N (for noun). A PUSH NP/ arc permits a transition if the state NP/ can recognize a noun phrase at the current spot in the input string. In addition, there are JUMP arcs which perform actions without advancing the input string (normally the input string is advanced past the word or words which permit a transition) and a variety of other special arc types. These are covered more fully in the appendices.

Figure 2-2 gives a simple example of a transition network grammar. It recognizes simple declarative and interrogative sentences with noun phrases containing adjective modifiers and prepositional phrases. Lexical arcs are indicated with lower case labels, and PUSH arcs are indicated with upper case labels that name the state to which control is to "push". It is easy to visualize the range of acceptable sentences from inspection of the transition network. To recognize the sentence, "Did the red barn collapse," the network is started in state S. The first transition is the aux transition to state q_2 permitted by the auxiliary "did". From state q_2 we see that we can get to state q_3 if the next "thing" in the input string is a NP. To ascertain if this is the case, we call the state NP. From state NP we can follow the arc labeled DET to state q_6 because of the determiner "the". From here, the adjective "red" causes a loop which returns to state q_6, and the subsequent noun "barn" causes a transition to state q_7. Since state q_7 is a final state, it is possible to "pop up" from the NP computation and continue the computation of the top level S beginning in state q_3 which is at the end of the NP arc. From q_3 the verb "collapse" permits a transition to the state q_4, and since this state is final and "collapse" is the last word the string is accepted as a sentence.



S is the start state

q_4, q_5, q_7, q_8, and q_{10} are the final states

Figure 2-2. A Sample Transition Network

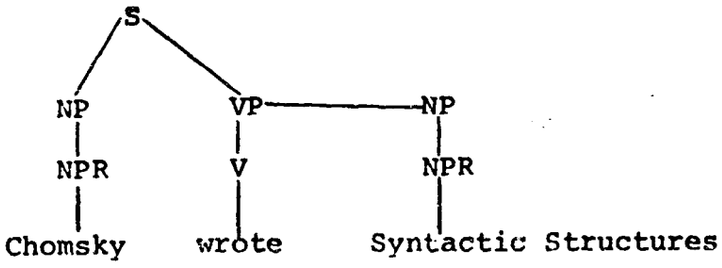
2.2.1 Structure building on the arcs

A parsing system must do more than just say whether or not a given string is a sentence; it must also build up a representation of the syntactic structure of the sentence. Such a representation must exhibit the syntactic relationships among the words and phrases of the sentence. In the augmented transition network model, this is accomplished by the use of structure-building actions on the arcs of the grammar, and by the association of a form with each final state of the grammar which specifies how to build the structural representation to be returned by that state. This form is given by the label on the POP arc associated with that state.

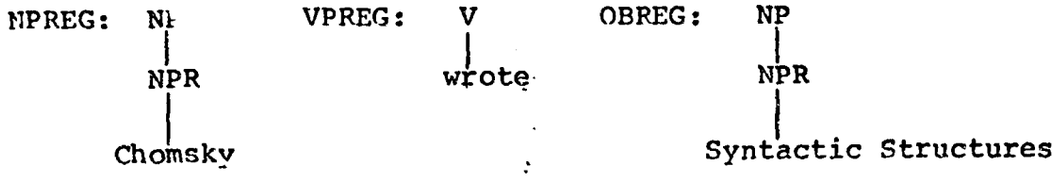
The structure-building actions as well as the arbitrary conditions on the arcs operate on the contents of a set of registers which are maintained at each level of recursive application of the network and are set and reset by the actions on the arcs. A special "current constituent pointer" * is also available for reference in the conditions and actions. The structure-building form associated with the POP arcs uses the contents of these registers to assemble its structural representation. Each register may contain an arbitrary piece of tree structure, and may also be used to hold flags for testing by the conditions on the arcs.

The basic structure building actions is that which attaches the contents of specified existing registers at specially marked points in a prototype tree fragment and puts the result into

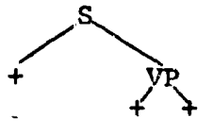
a register. For example, the sentence structure



can be built by attaching the contents of registers NPREG, VPREG and OBREG:



as the leaves of the fragment:



where the + signs indicate leaves that are to be replaced by register contents.

The use of registers to hold peices of sentence structure allows considerable flexibility in the way that structures are built up. The final structure of a construction does not need to be fixed until the parser is ready to pop up with the total structure of the construction. That is, the decision as to the final structure can be postponed until all of the pieces of structure that are found in various orders in the construction. The relative order among the

pieces of structure contained in different registers is not decided until the pieces are put together at the end, and this order need not have anything to do with the order in which the pieces were found. Moreover, even when one has made a tentative decision as to the funciton of a particular part of the structure and assigned it to a register accordingly, it is always possible to change one's mind in the light of subsequent input and move that piece of structure to a different register. Nothing about the structure is frozen until the moment that it is popped up to the higher level computation which wanted it.

2.2.2 Parsing with a transition network grammar

A transition network grammar is essentially a non-deterministic machine. That is, the transitions which are permissable from a given state are not uniquely determined by the input string. It is this characteristic of the model which mirrors the notion of ambiguity in English sentences. A sentence is ambiguous if there is more than one possible accepting path for that sentence. There are a number of complexities forced on a natural language parser by the fundamental ambiguity of English, and one of them is the need to provide an algorithm which is capable of pursuing various possible alternatives in the course of parsing, and the enumeration of these alternatives is the major source of effort in most natural language parsing systems. While it is not possible to avoid completely this fundamental fact of life for natural language processing, the techniques of the transition network grammar go a long way toward minimizing the problem. The factoring and merging of paths in the network and the postponing of decisions until there is information to make them tend to reduce the total number of alternatives which in principle must be considered. Furthermore, the ordering of the arcs leaving the states permits a selection of the "more likely"

alternatives first so that in many cases, the most likely parsing is found while many of the other alternatives have not yet been persued. This permits a parsing system in which the parser uses the ordering of the arcs and the complex conditions on the arcs to try to determine the most likely parsing first and thereby avoid a large part of the enumeration required by other parsing algorithms. The basic necessity for dealing with a non-deterministic or enumerative algorithm, however, remains.

2.2.3 Configurations

In simulating the operation of a nondeterministic machine by a deterministic machine such as a real computer, it is necessary to keep track of alternative configurations of the nondeterministic machine. In the case of a transition network grammar a configuration is determined by the current state, the current register contents, and a stack of the states and register contents at all higher levels in the analysis (since in general, the current state may be several levels down in recursive calls to the network). Each recursive call to the network adds another entry to the stack to contain the state and registers associated with that level and then clears the registers for the new level and sets the state to the state which was named on the PUSH arc. In addition, the stack entries remember the actions which remain to be performed on the PUSH arc after a successful return form the PUSH.

In the parsing system which we have implemented, the configuration is represented by a list consisting of the state, the stack, a list of register contents, the contents of a special HOLD list, and a PATH entry which records the history of how the current state was reached from the initial state at the current level. The stack is represented by a list whose elements (STACKELT's) record

the state, the register contents, the actions on the PUSH arc, and the partial path entry for the computations at higher levels. Register contents are kept on a list of alternating register names and register values.

2.2.4 Organization of the parser

Parsing of a sentence begins by calling the function PARSER with a string to be parsed. PARSER constructs an initial configuration consisting of the start state (with empty registers and stack) and then calls a function STEP to simulate the transitions in the network. It calls a function LEXIC to perform the lexical analysis of the input string--determining the next word, accessing its dictionary entry, expanding contractions, compressing compound expressions, making substitutions, etc. Thus PARSER provides the basic overall control, while LEXIC interfaces the input string and STEP performs the basic simulation of the transition network. Flow charts of these basic functions are given in figures 2-3, 2-4, and 2-5.

2.2.5 Simulation of Nondeterminism

Although the parsing system we are using provides for the following alternative paths either in series or in parallel or in combinations of the two, the DDC system as we have implemented it makes use only of the sequential mechanism. In this mode, the arcs leaving a state are considered in the order in which they occur, and the first arc which can be followed is chosen. At this point, any arcs remaining in the list, together with the current configuration and the place in the input string, are combined into a list called an ALTARC alternative and saved on an ALTS list of the parser to be persued later if the current choice turns out not to be successful.

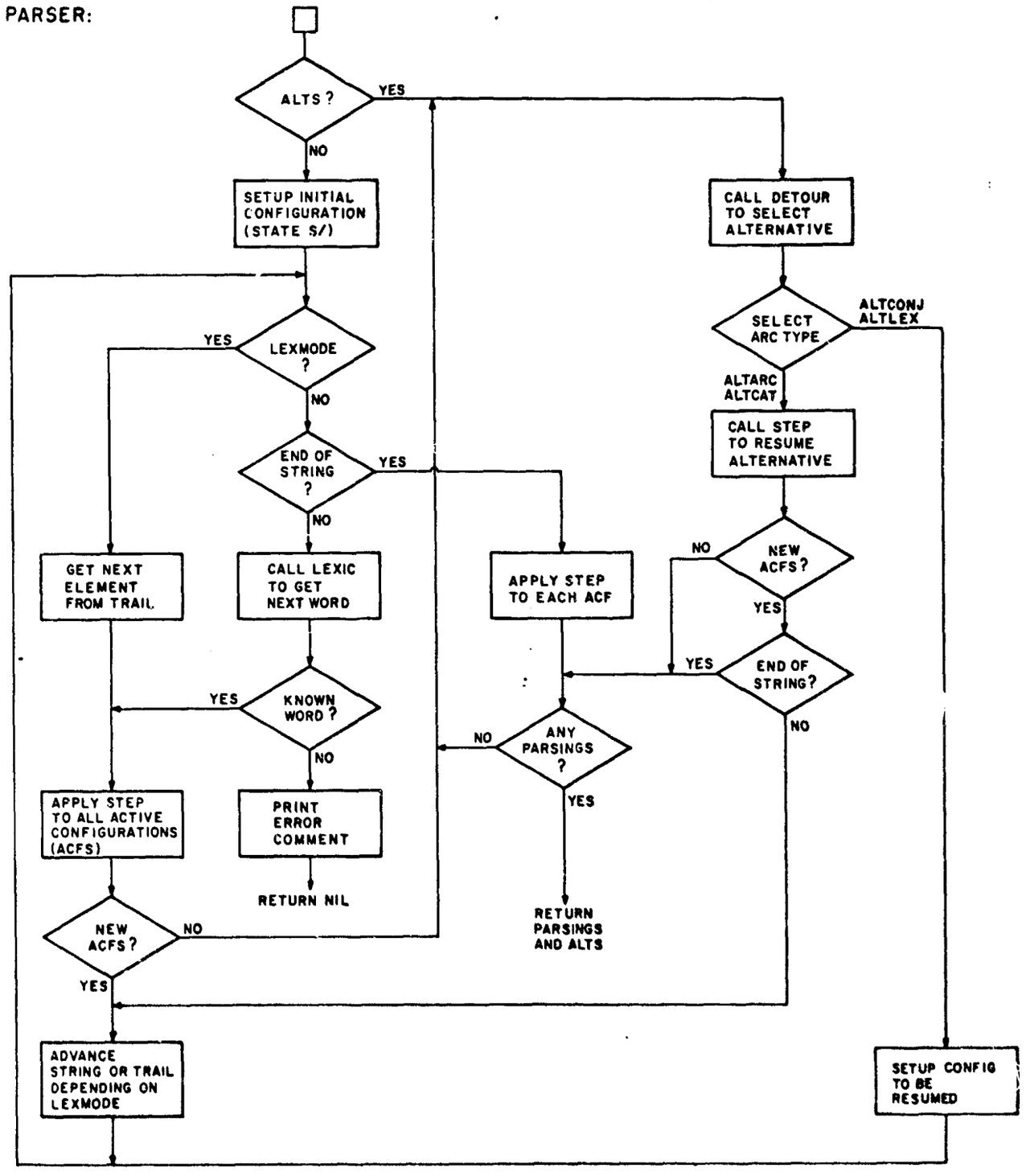


Figure 2-3. The Function PARSER

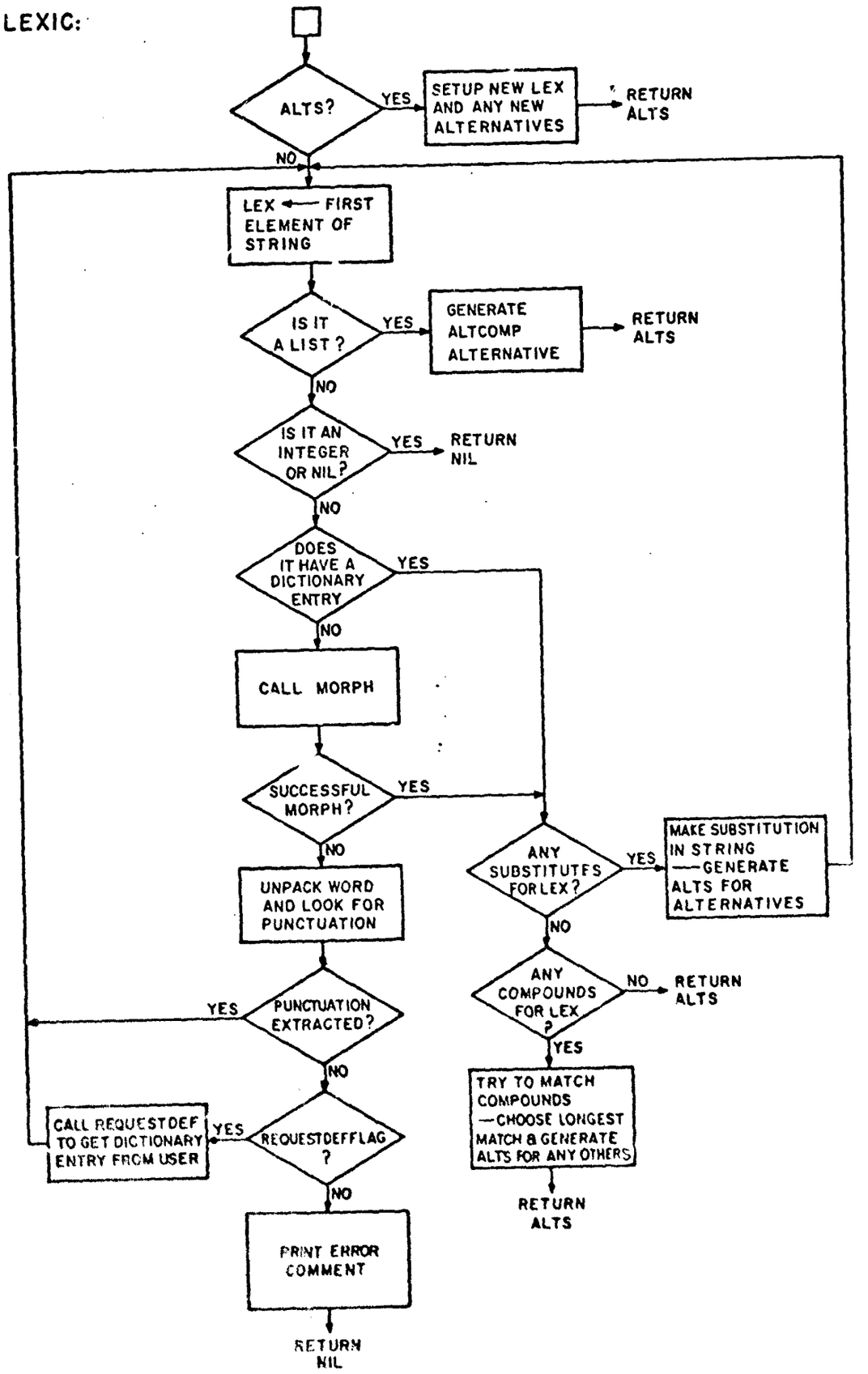


Figure 2-4. The Function LEXIC

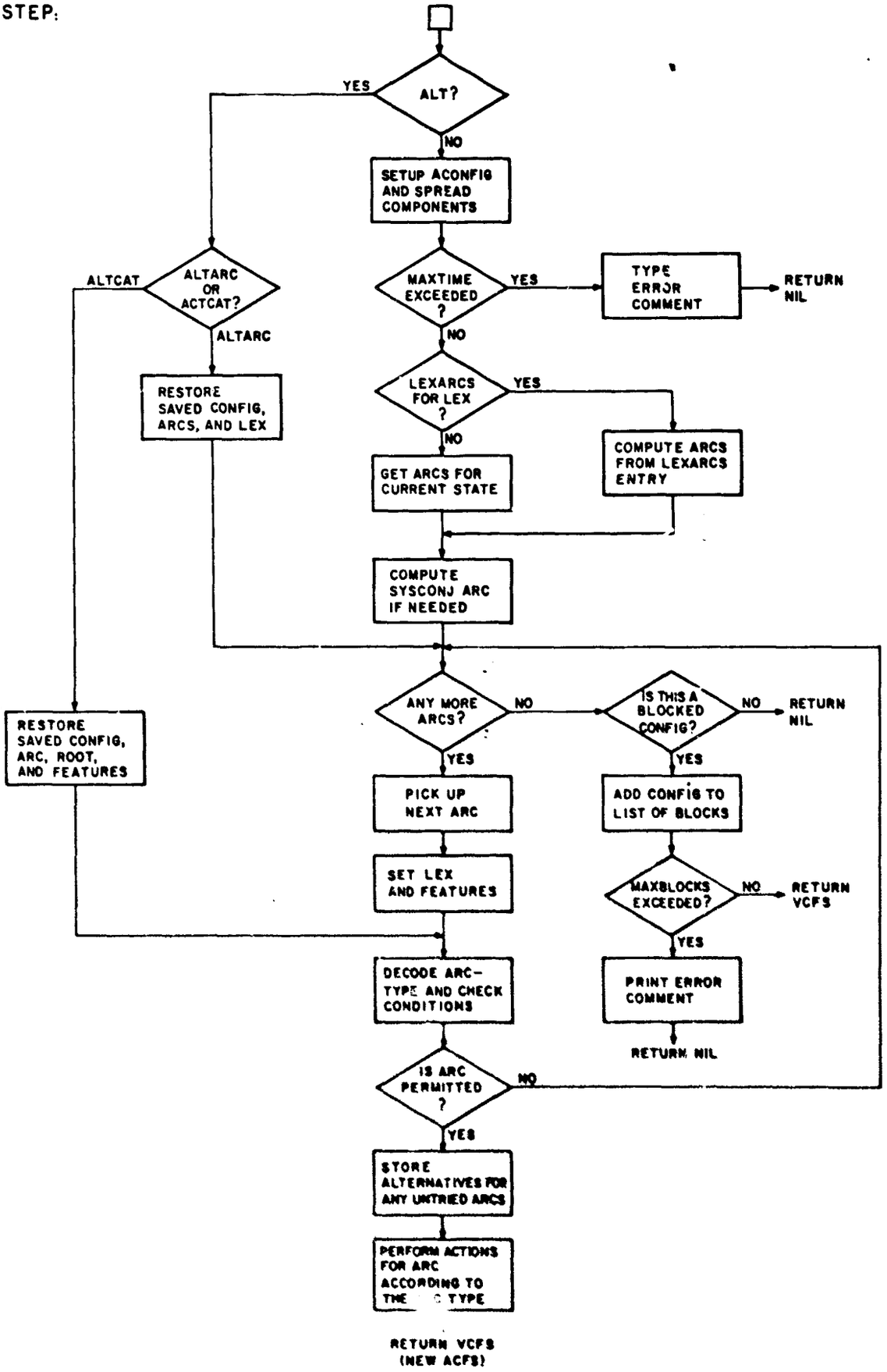


Figure 2-5. The Function STEP

We feel that this means of dealing with nondeterminism preserves the potential for backtracking and trying other alternatives which is essential for dealing with natural language ambiguity while retaining most of the advantages of a deterministic algorithm. It depends for its success, however, on the ability for selecting the right parsing first or soon thereafter, (since otherwise all alternatives have to be enumerated and the advantages are lost). The present grammar does a very good job of selecting a reasonable parsing (if not the best one) in most cases, but there remains one major area in which syntactic information alone (without semantic information) has so far proven insufficient for making good choices for the "most likely" parsing. This area is that of choosing the most likely scope for reduced conjunctions. This problem will be discussed more thoroughly in Chapter 3.

2.2.6 Morphological analysis

One of the features of the current English Preprocessor is a facility for morphological analysis of regularly inflected nouns and verbs. This facility permits a single dictionary entry for the root form of the word with a code which indicates the type of regular inflection which the word undergoes. The system will then automatically recognize all of the regularly inflected forms of that root. This facility is performed by a function MORPH called by LEXIC. In addition to inflectional analysis, MORPH is able to recognize some items which appear to be contract numbers, hyphenated adjective modifiers, integers, and times of day without their having to be entered in the dictionary. Other types of morphological analysis are possible for "guessing" the parts of speech for words that are unknown to the system, but this type of analysis has not been incorporated into the current system.

2.3 The Semantic Interpreter

The semantic interpretation component of the DDC English preprocessor is an adaptation of the semantic interpretation procedure presented in Woods (1967, 1968). It operates on a syntactic structure or fragment of syntactic structure which has been constructed by the parser and it assigns semantic interpretations to the nodes of this structure to indicate the "meanings" of those constructions to the system. The procedure is such that the interpretation of nodes can be initiated in any order, but if the interpretation of a node requires the interpretation of a constituent node, then the interpretation of that constituent node is performed before the interpretation of the higher node is completed. Thus, it is possible to perform the entire semantic interpretation by calling for the interpretation of the top node (the sentence as a whole), and this is the normal mode in which the interpreter is operated in the DDC system. In other applications of natural language processing, people have argued for the interpretation of constituents from the bottom up as they are built by the parser in order to rule out semantically bad parsings early in the process. This argument does not apply to the DDC situation, however, since the system does not contain sufficient semantic information to make such judgements.

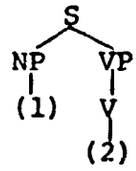
2.3.1 Semantic Rules

In determining the meaning of a construction, two types of information are used--syntactic information about sentence construction and semantic information about constituents. For example, in interpreting the meaning of the sentence, "Chomsky wrote Syntactic Structures," it is both the syntactic structure of the sentence (subject = Chomsky; verb = "write"; object = Syntactic Structures) plus the semantic facts that Chomsky is a person and Syntactic

Structures is a book that determine the interpretation (AUTHOR: SYNTACTIC STRUCTURES CHOMSKY). In the Woods interpretation procedure, this information is embodied in semantic rules consisting of patterns that determine whether a rule can apply, and actions that specify how the semantic interpretation is to be constructed.

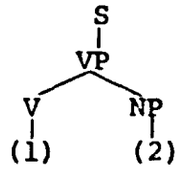
Syntactic information about a construction being interpreted is tested by tree fragments such as those indicated below:

S:NP-V



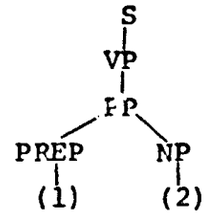
SUBJECT-VERB

S:V-OBJ



VERB-OBJECT

S:PP



PREPOSITION-OBJECT
MODIFYING A VP

Fragment S:NP-V matches a sentence if it has a subject and a verb and also associates the numbers 1 and 2 with the subject noun phrase and the verb respectively. The numbered nodes can be referred to for checking semantic conditions and for specifying the interpretation of the construction. Fragments in the system are named mnemonically for readability.

The basic element of the pattern part of a semantic rule is a template consisting of a tree fragment plus additional semantic conditions on the numbered nodes of the fragment. For example, the template (S:NP-V (AND (MEM 1 PERSON) (EQU 2 WRITE))) matches a sentence if its subject is semantically marked as a person and its verb is "write". The pattern part of a rule consists of a limited

Boolean combination of such templates and the action of the rule specifies how the interpretation of the sentence is to be constructed from the interpretations of the numbered nodes of the templates.

The left-hand side of a semantic rule consists of a list of components, each of which may be either a single template, a negated template (embedded in a NOT), or a disjunction (OR) of templates. A component consisting of a simple template matches a node of the syntax tree it its template does, and NOT component matches a node if its embedded template fails. An OR component matches if any of its constituent templates match (including a possible DEFAULT template at the end which matches if nothing else does). A semantic rule matches a node if all of its components match. In addition, in the process of matching a rule, a record is maintained of the nodes of the syntax tree which match the numbered fragments in each of the components.

2.3.2 Right-hand Sides

The right hand sides (or actions) of semantic rules are forms (or schemata) into which the interpretations of embedded constituents are inserted before the form is evaluated to give the semantic interpretation (or a part of it) which is to be attached to a node. The expressions in the right-hand sides which indicate the places where interpretations of embedded constituents are to be inserted are indicated by lists (called a REF's) which begin with the atom # and contain one or two numbers and an optional "TYPEFLAG". The numbers indicate the node in the tree whose interpretation is to be inserted by naming first the number of a component of the rule and then the number of a node in a tree fragment of that component. Thus the reference (#2 1) represents the interpretation of the node that matches node 1 of 2nd component of the

rule. In addition, the single number ∅ can also be used to re-reference the current node.

The TYPEFLAG element, if present, indicates how the node is to be interpreted. (For example, in the DDC system there is a distinction between interpreting a node as a topic description and interpreting it for what it says.) Thus (# ∅ TOPIC) represents the interpretation of the current node as a topic description. There are a variety of types of interpretation used for various purposes in the semantic interpretation rules of the system. The absence of a specific TYPEFLAG in a REF indicates that the interpretation is to be done in the normal mode for the type of node which it matches. In this case, there is an alternative form of the REF consisting of a dotted pair of the two numbers. Thus (2 . 1) is equivalent to (# 2 1).

As an example, consider the semantic rule:

```
(S:WRITE
     (S:NP (MEM 1 PERSON))
     (S:V-OBJ (AND (MEM 2 DOCUMENT) (EQU 1 WRITE)))
        -- (PRED (AUTHOR: (# 2 2) (# 1 1))))
```

This rule says that if the sentence has a subject which is a person, a verb "write", and an object which is a document, then the meaning of the sentence is computed by substituting the interpretations of the node numbered 1 in the first component (#1 1) and the node numbered 2 in the second component (#2 2) into the indicated places in the schema (AUTHOR (#2 2) (#1 1)) and treating it as a predicate (PRED). (S:WRITE is the name of the rule.)

2.3.3 Organization of Rules

The semantic rules for interpreting sentences are usually governed by the verb of the sentence. That is to say that out of the entire set of semantic rules, only a relatively small number of them can possibly apply to a given sentence because of the verb mentioned in the rule. For this reason, the semantic rules can be indexed according to the verb (or verbs) of sentences to which they could apply and recorded in the dictionary entry for the verb. Each rule then characterizes a syntactic/semantic environment in which the verb can occur and specifies its interpretation in that environment. The templates of the rule thus describe the necessary and sufficient constituents and semantic restrictions in order for the verb to be meaningful. There are also situations, however, in which the type of construction and the mode in which it is being interpreted determine a set of rules which does not depend on the head of the construction.

2.3.4 Multiple Matches

Since the templates of a rule may match a node in several ways, and since several rules may simultaneously match a single node, it is necessary to indicate how the interpretation of a node is to be constructed in such a case. To provide this information, the lists of rules which the interpreter uses--whether taken form global lists or from the property lists of heads of constructions--are not necessarily simple lists of rules, but may be organized into rule groups with each group indicating how (or whether) simultaneous matches by different rules are to be combined. In addition, at top level of such lists, the atom NIL may be used as a "barrier" to indicate that by the time the matching process has reached that point in the list it will proceed further only if there have been no successful matches so far.

The mode for combining simultaneous matches at the top level of this list is a default mode determined by TYPEFLAG and the type of node. Possible modes are SPLIT (which keeps multiple matches separate as semantic ambiguities), FAIL (which prohibits multiple matches), AND (which combines multiple matches with an AND), and OR (which combines multiple matches with an OR). For example, a rule list of the form (A B NIL C (OR D E)) with default mode AND indicates that if either of the rules A or B is successful, then no further matches are tried (NIL is a barrier); otherwise, rules C, D, and E are tried, and if both D and E match then the results are OR'ed together, and if C matches together with D or E or both, it is AND'ed to the results of the OR group.

The modes (SPLIT, FAIL, AND, and OR) also apply to multiple matches of a single rule. A rule may either specify the mode for multiple matches as its first element prior to the list of components, or else it will be governed by the rule group mode setting at the time it is matched.

2.3.5 Organization of the Semantic Interpreter

The overall operation of the semantic interpreter is as follows: A top level routine calls the recursive function INTERP with TYPEFLAG NIL looking at the top level of the parse tree. Thereafter, INTERP attempts to match semantic rules against the specified node of the tree, and the right-hand sides of matching rules specify the interpretation to be given to the nodes. The possibility of semantic ambiguity is recognized, and therefore the routine INTERP produces a list of possible interpretations (usually a singleton, however). Each interpretation consists of two parts--a node interpretation (called the SEM of the node) and a quantifier "collar" (called the QUANT of the node) which is to be returned to

the routine which called for the semantic interpretation of the current node. Thus the result of a call to INTERP for a given node P is a list of SEM-QUANT (or S-Q) pairs--one for each possible interpretation of the node.

INTERP then calls a function HEAD to determine the head of the construction which it is interpreting and a function RULES to determine the list of semantic rules (depending on the type of node and the value of TYPEFLAG) which it is to use to interpret the construction. It then dispatches control to a routine MATCHER, then, depending on the TYPEFLAG and various mode settings, INTERP either returns a default interpretation T, goes into a break with a comment that the node is uninterpretable (permitting a systems programmar to debug rules), or returns NIL indicating that the node has no interpretations for the indicated TYPEFLAG.

The function MATCHER calls a function MATCHGROUP to match groups of semantic rules, and MATCHGROUP, in turn, calls the function RMATCH to match single rules. RMATCH calls the function TEMPMATCH to match templates in the left-hand side of the rule and SEMSUB to insert the interpretations of constituents into the right-hand side of the rule and compute the resulting interpretation. The relationships among these functions is indicated by the diagram in Figure 2-6, and flowcharts for the routines INTERP and RMATCH are given in figures 2-7 and 2-8.

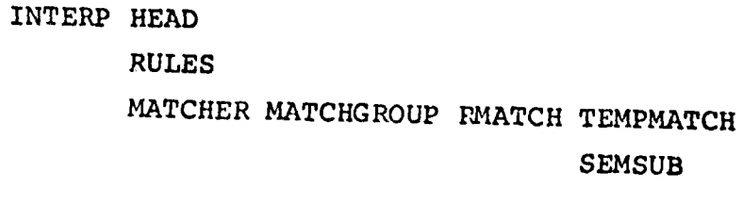


Figure 2-6. Subroutine control map for the routine INTERP

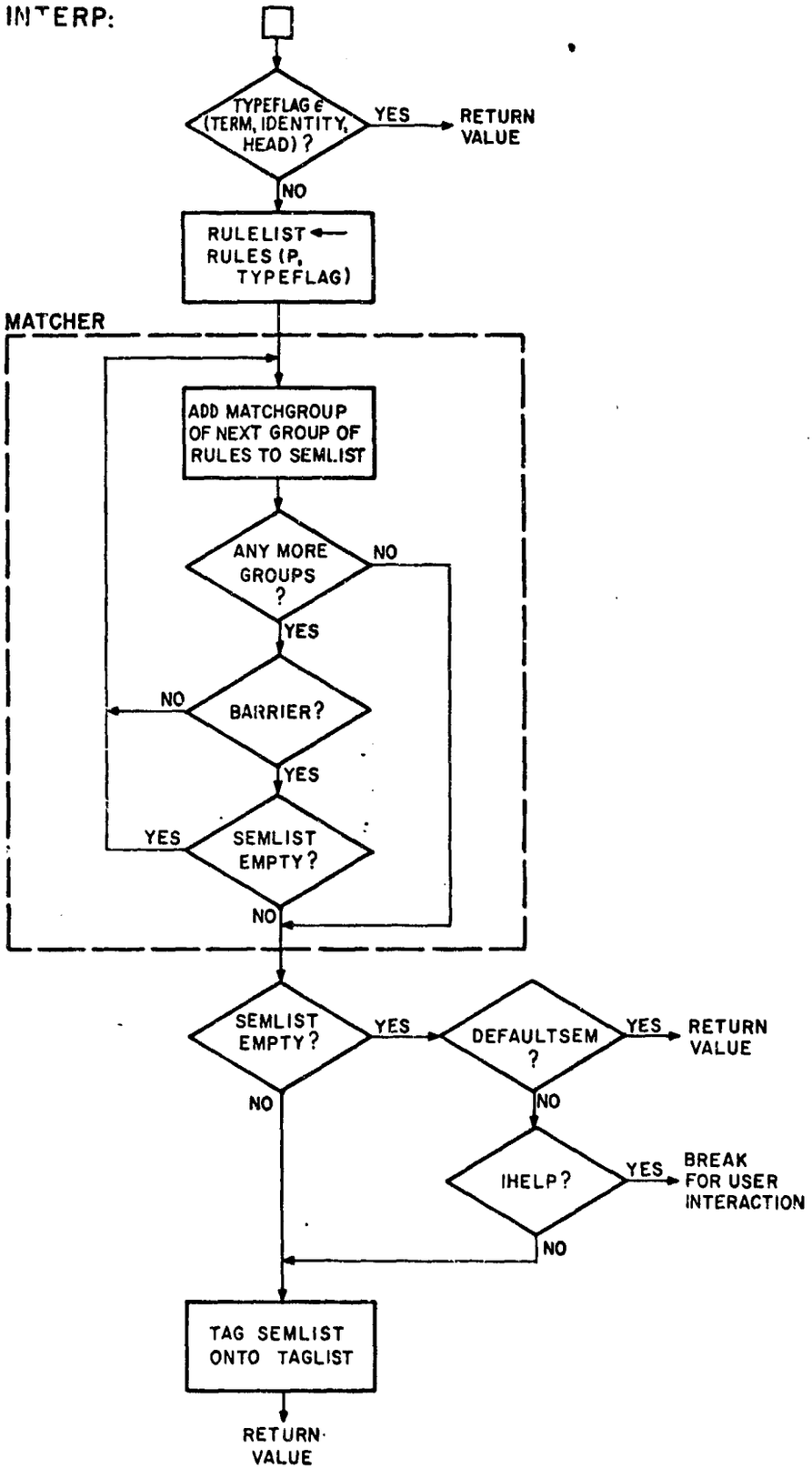


Figure 2-7. The Function INTERP

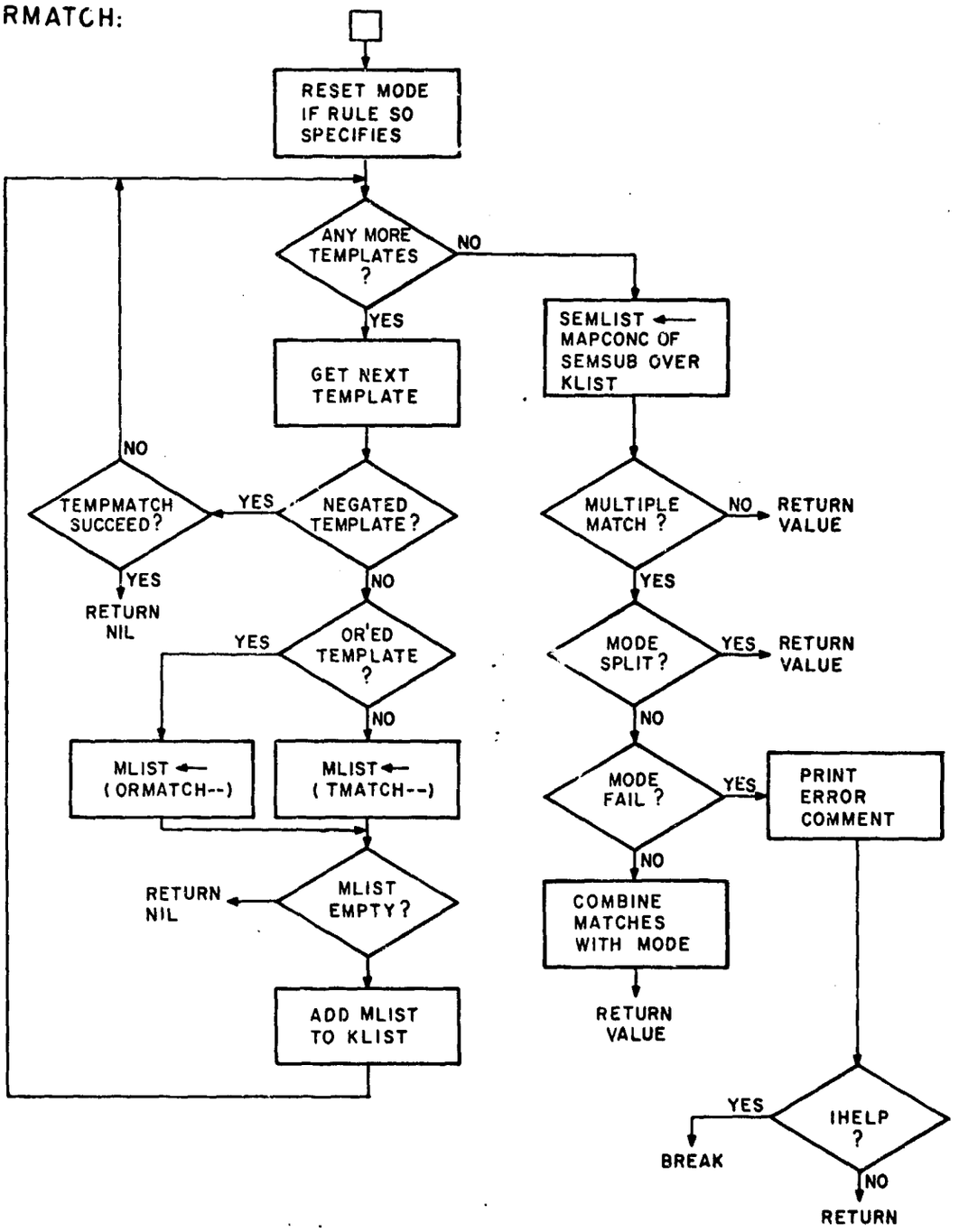


Figure 2-8. The Function RMATCH

2.4 Boolean Request Generation

The semantic interpretations assigned to sentences by the system are represented in a powerful, general purpose semantic query language (described more fully in Woods, 1967, 1968). Commands in the language may be quantified by quantifiers of the form (FOR QUANT X/ CLASS : P(X) ; Q(X)), where QUANT is a quantifier (such as EACH or SOME), X is the variable of quantification, CLASS is the class of objects over which quantification is to range, and P(X) is a restriction on the range of quantification which selects the set of objects of interest. The quantified command Q(X) indicates the operation which is to be performed for the selected objects and may itself be quantified (as can the restriction P(X)). Because of its facility for multiple quantification over a number of files, the semantic query language is capable of representing a range of query tasks of much larger scope than the present DDC retrieval system. Formally it has the complete expressive power of the first order functional calculus. We consider this an important advantage since it provides a semantic framework which will continue to be valid for extended retrieval facilities and additional types of service which may be envisaged for the future. More importantly for the immediate future, this extended query language will permit the system to interpret requests which are beyond the scope of the current system and to provide the user with explicit, useful diagnostics explaining why the request cannot be processed and perhaps suggesting alternative requests which would provide equivalent information. Also, the semantic query language suggests a number of additional facilities which could be provided for an interactive English querying system using the current DDC data files with small additional programming effort.

2.4.1 Extracting the Boolean Request

The Boolean request generator performs the task of examining a semantic representation, determining if it falls within the range of things accepted by the current system, and generating a conjunctive normal form Boolean combination which approximates the input language of the DDC on line retrieval system. The range of things accepted are currently limited to a single quantification over a set of documents with a restriction on the range of quantification which consists of a Boolean combination of instances of the predicate ABOUT (which asserts that a document is about a topic). This is handled in the current system by defining FOR as a LISP function which checks that the class over which the quantification is to range is DOCUMENT, and that there is at least one instance of the predicate ABOUT in the request (printing appropriate comments if such is not the case). It then calls a function BOOLGET to extract the topics from all instances of the predicate ABOUT from the condition P(X) and combine them in a Boolean expression which corresponds to the way in which they were combined in P(X). In all normal cases, P(X) is simply a Boolean combination of instances of ABOUT, and BOOLGET is merely the corresponding Boolean combination of topics (but the topics themselves may also be arbitrary Boolean combinations of key phrases).

The Boolean request extracted by BOOLRET is then converted to conjunctive normal form by a routine CNF, adjusted into a form approximating the DDC syntax by BOOLREQ, and printed out to the user by the function BOOLRET. In a complete system, BOOLRET would

be the function which would actually pass the Boolean request to the data base and print out the results of the search.

2.4.2 Conjunc ve Normal Forms

The DDC remote on-line retrieval system (hereafter abbreviated RORS) whose input syntax was taken as a pattern for the output of the English preprocessor provides for search patterns consisting of a conjunction of "levels," each of which may be either a single term or a disjunction of terms, and an optional exclusion level, which may presumably be either a single term or a disjunction of terms. A BNF specification of the possible patterns is:

<pattern> → <conjunct> NOT <"level"> | <conjunct>
<conjunct> → <conjunct> AND <"level"> | <"level">
<"level"> → <"level"> OR <term> | <term>

Thus the most general type of question one can phrase within this language has the form:

a_1 OR a_2 OR

AND

b_1 OR b_2 OR . . .

AND

.
.
.

x_1 OR x_2 OR . . .

NOT

y_1 OR y_2 OR . . .

That is, one can express arbitrary Boolean combinations containing no negations and one can exclude any number of single terms, but one cannot exclude a conjunction of terms. That is, one cannot exclude a term A only when it cooccurs with term B, but must either exclude it or keep it in its entirety. A logical expression of the form $A \wedge B \wedge \neg (C \wedge D)$ cannot be represented in a RORS search pattern.

An example of such a sentence is:

"Give me documents on information retrieval which are not about relevance and measures."

Such constructions are somewhat unlikely to occur directly as requests, but are more likely when a user negates an English phrase which expands into a conjunction during the Boolean key phrase extraction. In order to avoid any potential difficulty with such constructions, we are currently using a Boolean query notation which allows the negation of a conjunction as well as a single term.

The restrictions of the RORS syntax were apparently designed to eliminate the unreasonable Boolean combinations which can be constructed using negations that would involve the complement of an inverted file (i.e. the list of all references are not in the file). A simple example of such an unreasonable request would be "Give me all references which are not about Information

Retireval". All of the requests that one would like to rule out as unreasonable have this characteristic that they return virtually everything in the database. Specifically they are the complements (or negations) of the reasonable requests. However, it turns out that there are valid and reasonable requests which cannot be expressed in the RORS syntax. Therefore we have adopted a syntax which is effectively an extension of the RORS syntax. Like the RORS syntax, it is a variation on the form known to logicians as conjunctive normal form, which consists of a conjunction (AND) of components each of which is a disjunction (OR) of terms or negations of terms, and the computation of a logical conjunctive normal form is the first step in constructing our representation.

The disadvantage of the normal logical notation for conjunctive normal form when applied to a Boolean retrieval system is that the negations in it are forced down to the lowest levels inside OR's. This is no problem when the negated term is the only component of the OR, since the negation is then essentially a component of the top level AND, and can be handled by merely subtracting off the documents which it retireves from those obtained by intersecting the other components of the AND. When the negated term is really a component of an OR, however, it causes problems, since the normal procedure for handling a Boolean retrieval of an OR'ed expression is to take the union of the lists of references retrieved by the components. When one of the components is negated, the list of references retrieved is virtually all of the collection and so the union will be also, and all of the advantages of operating with inverted files would be lost. The RORS syntax avoids this problem by simply forbidding NOT's within OR's, but this is overly restrictive.

The Boolean request syntax that we have used adopts the following simple strategy when the conjunctive normal form representation of the query contains a negated term:

1. If there is no other component of the OR, then the negated term is really a component of the AND and no change is necessary.

2. Otherwise, raise the negation by changing the OR to an AND embedded in a NOT and complementing each of its components (This is simply an application of the well known DeMorgan's law).

After application of this strategy, all NOT's will be embedded within AND's, and the request will be reasonable if there is some unnegated term in the top level AND; otherwise, unreasonable. To make the effect of this transformation more clear, we carry it one step further and group the components of an AND into two groups--those terms which are negated and those terms which are not. If there are some terms of each type, then we replace the AND with an instance of a connective SDIFF whose first argument is the AND of the unnegated components, and whose second argument is the OR of the negated terms (with their negations removed). SDIFF (for set difference) is a function which can operate efficiently with inverted files to subtract elements of the second list from those of the first. In this final form, an expression is unreasonable if and only if it is a negation at the top level (in which case it is the negation of a reasonable request). An example of a request containing SDIFF's would be:

```
(SDIFF (AND (WATER)
            (POLLUTION))
       (ORGANIC))
```

corresponding to a request for "non-organic water pollution". Expressions which contain no SDIFF or at most a single SDIFF as the topmost connective are representable in the RORS syntax (and indeed differ from the RORS syntax in only trivial ways). Expressions containing more than one SDIFF are generated only when the Boolean request is not representable within the RORS syntax.

Chapter 3

THE GRAMMAR

The translation of an English document request into an appropriate Boolean expression proceeds in three main stages: first, the English sentence is converted into a "canonical form" in which the syntactic relationships holding between constituents are made explicit; next, the canonical form, or parse, is mapped into a semantic interpretation which highlights the logical connections between terms; and finally, the semantic interpretation is reduced to a conjunctive-normal-form Boolean combination of document descriptors. In this section we discuss in some detail the first stage of the translation process, the parsing of the input string of English words.

3.1 Motivation and Overview

It is a well-known fact about natural languages that sentences which have different words in different orders can have essentially the same meanings, while superficially similar sentences can have very different meanings; this insight is the cornerstone of the transformational theory of grammar (Chomsky, 1957, 1965). For example, active sentences (1) have corresponding passive sentences (2) which are virtually synonymous, and sentences with existential "there" subjects (3) are synonymous to sentences with ordinary subjects (4):

(1) We need some information.
(2) Some information is needed by us.
(3) There are many documents in the file.
(4) Many documents are in the file.

On the other hand, sentences (5) and (6) have similar sequences of "parts-of-speech", but the syntactic and logical relationships between the words are different: in (5), John is to do the pleasing, whereas in (6) John is to be pleased by someone. Linguists account for these facts by positing

(5) John is eager to please.
(6) John is easy to please.

a form of syntactic description more abstract than just a specification of the linear arrangements of words in sentences. In brief, transformational grammarians characterize the syntactic relationships in a sentence in terms of a "deep structure", a structural description, usually in the form of a tree with labeled nodes, from which the string of words can be derived by applying a sequence of formal rules called transformations (Chomsky, 1965). Thus the similarity in meaning of actives and their corresponding passives is due to the fact that the sentences have the same deep structure; the different strings result from the application of slightly different sequences of transformations. Sentences with similar superficial characteristics but different syntactic relationships, such as (5) and (6), result from the application to different deep structures of transformation sequences with similar outputs. Finally, ambiguous sentences can be derived from more than one deep structure, while ungrammatical word-strings have no corresponding deep structure. In this context, the major task in the syntactic analysis of an input document request may be seen as the problem of determining the appropriate deep structure (s) for a given string of English words.

Unfortunately, linguistic theory is not of much help here. Transformational grammars are designed to enumerate the class of possible deep structures (by using a context-free phase-structure grammar) and then to generate all and only the sentences of a language from the set of deep structures. Very little has been said about how to find the deep structure for a given string, and, in fact, the few attempts at "reversing" the sequence of transformational operations that have been made have not been very successful (cf. Petrick, 1956; Zwicky, et al. 1965). Thus, for the syntactic analysis component of the DDC English preprocessor we have used an augmented recursive transition network parser, described elsewhere in this report and in Woods (1970), which surmounts many of the difficulties encountered in earlier efforts at transformational recognition. Our goal is still to map the input request into a deep-structure-like representation.

A transition network grammar consists of a set of states connected by a set of labelled directed arcs. The label on an arc determines whether the transition can be made, based on the current input word and the previous analysis history of the input string, and also specifies a set of actions to be executed if the transition is permitted (section 2.2 of this report gives a detailed description of the grammatical notation and the operation of the parser). The sequence of transitions taken in the course of an analysis reflects the superficial arrangment of words in the string; the actions on the arcs are used to build up sections of the deep-structure tree and hold them in "registers" until they are combined into larger sections and, eventually, into the complete representation of the input string. Analysis of the input string thus proceeds from left to right on two related levels: words in the string are identified by arc transitions, and at the same time, the deep-structure is being fashioned in the registers.

The configuration of arcs and states in the grammar is shown in Figure 3-1. Unless the order of the arcs is explicitly indicated by numbers on the arcs, they are ordered clockwise from the top of the state. The symbol & on an arc indicates that there is a condition associated with the arc which is not included in the figure. See the grammar listing in Appendix B for the details of these conditions.

The parses developed by the grammar resemble the deep-structures described by Chomsky (1965), with some elements borrowed from Stockwell et al. (1968) and some included because of special characteristics of the DDC document requests. Briefly, a sentence consists of a subject noun-phrase, an auxiliary-verb constituent specifying the tense, modality, and aspect of the sentence, and a verb-phrase containing the main verb, the direct and indirect objects and predicate complements (if any), and optional adverbial and prepositional-phrase modifiers. A noun-phrase consists of an optional determiner and adjectival modifiers, a head noun, and optional restrictive and non-restrictive post-nominal modifiers. A precise specification of the form of deep-structures is contained in the listings of the grammar actions SBUILD, NPBUILD, and DETBUILD in Appendix D and in the annotated listing of the grammar itself (Appendix B). Below we will focus on the grammatical strategies used to identify the various constituents and not on the structures in which they are placed.

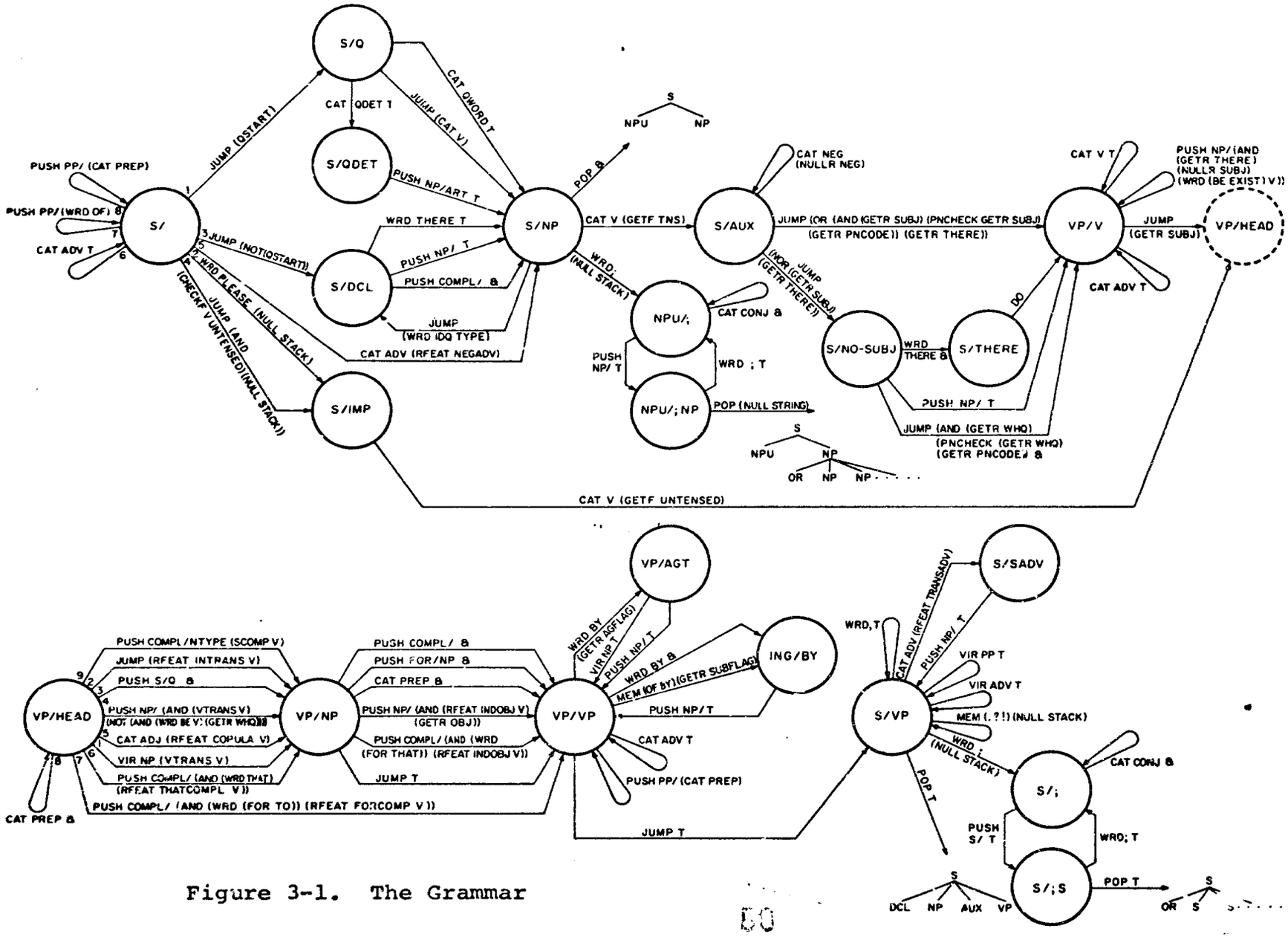


Figure 3-1. The Grammar

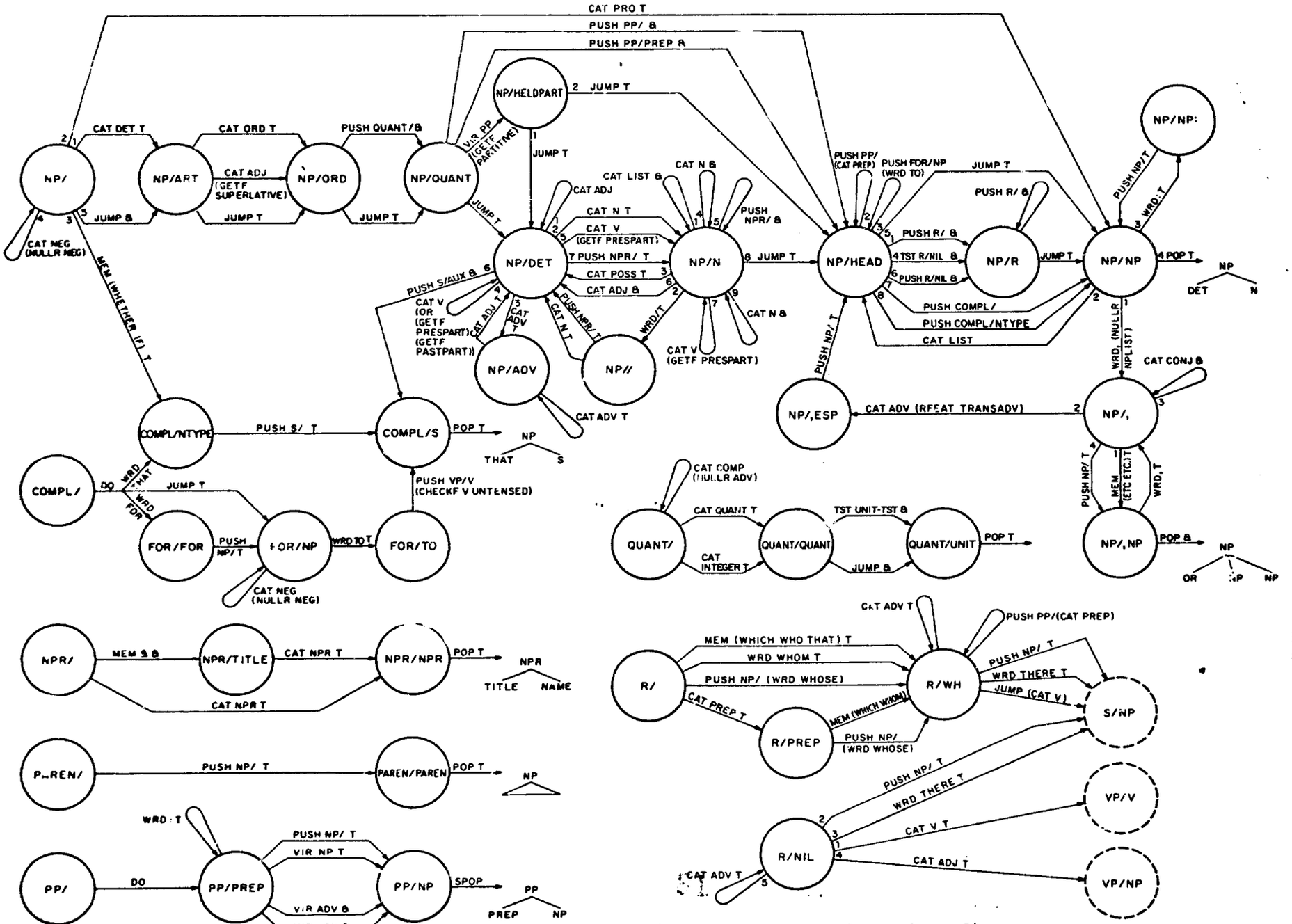


Figure 3-1. (continued)

3.2 General Description of the Grammatical Strategies

In this section we discuss and illustrate the overall organization of the grammar and indicate in some detail the strategies used to deal with particular syntactic constructions. A bird's-eye view of the grammar was given in Figure 3-1 in which states are represented by circles enclosing the state name (for example, S/DCL) and arcs are represented by arrows connecting the states. The arcs are labelled in the diagram with their types (CAT, WRD, MEM, VIR, JUMP etc.) and frequently, with their conditions also. The actions on the arcs and the detailed specification of complicated conditions may be found in the annotated listing in Appendix B.

The parser allows state names to be arbitrary LISP atoms, but we have adopted the convention that state-names indicate the unit of the sentence being analyzed and constituents of the unit already identified, separated by a slash ("/"). Thus S/AUX signifies that the S-level of the parse is being developed and that we have succeeded either in finding an auxiliary verb or in establishing the fact that the sentence has no auxiliary. The diagram also expresses another convention: unless the arcs leaving a state are explicitly numbered in the diagram, the clockwise order of arcs from the top of a state-circle corresponds to the order of arcs in the grammar listing and the order in which transitions are attempted. By convention, the initial state of the whole grammar is state S/.

3.2.1 The sentence level network

With these conventions established, we can examine the way in which the parser uses this grammar to analyze sentences:

1. The basic strategy

Consider the simple sentence (7):

(7) I need information.

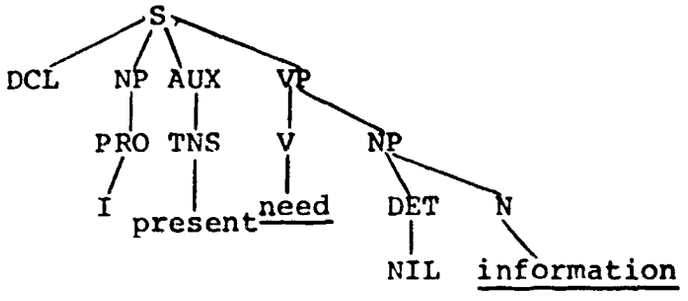
The underlying structure of (7) is intuitively obvious, given only a slight familiarity with high-school grammar: I is the subject, need is the verb, and information is the object. The parser begins by comparing the string to the grammar at state S/. The first word of the string (I) cannot start an English question, so the predicate QSTART fails, ruling out the first arc but permitting the third. Since I is not please, the second arc is also excluded, and so the third arc is the first transition. We jump to state S/DCL, having established that the sentence is declarative. Since a JUMP transition does not advance the input string, we are still looking at I. At S/DCL we try to find the subject noun-phrase, the word "there" in subject position, or a subject complement. In this case, the push to the noun-phrase network (arc 2) is successful, returning the structure (8).

(8)


We enter state S/NP with need as the current word and (8) in the register SUBJ. Since need is a tensed verb, the CAT V transition is permitted, and the actions save the tense (present), the person-number code (X35G = "anything except third-person singular"), and the root from of the verb (need) in the appropriate registers. We enter S/AUX looking at the last word of the sentence. Since we have already identified the subject and since its person and number agree with those of the verb, we jump to VP/V, and from there we jump to VP/HEAD. VP/HEAD is a landmark: whenever we reach it, we have identified the main verb (the head of the verb phrase) and the subject, and we can begin to look for post-verbal constituents. In this case, since need is transitive, we push

for the object noun-phrase at arc 3, and successfully return with the object, information. This is the end of the string, so we continue jumping through the grammar from VP/NP to VP/VP and then to S/VP, from which we pop the recovered deep structure (9):

(9)



This is the basic strategy for simple, active, declarative, transitive sentences: at S/DCL, we have decided that the sentence is declarative; at S/NP, we have the subject; at S/AUX, we have the first (and only) verb, which carries us through to VP/HEAD; at VP/NP we have the direct object, and we then jump all the way to S/VP, where we pop the completed parse. For intransitive sentences such as (9),

(9) I went.

the jump arc (arc 1) is taken from VP/HEAD instead of the PUSH NP/ arc, and the resulting structure does not have the NP node in the VP.

From this basic strategy, more complicated sentences are analyzed by varying and elaborating one or more segments of the analysis path.

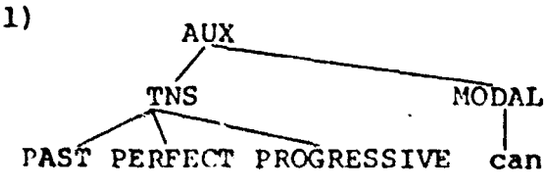
2. Auxiliary verbs

If the sentence has one or more auxiliary verbs beside the main verb, as in (10), the analysis path is embellished at

(10) I could have been going.

state VP/V: the CAT V arc at state S/VP picks up the modal verb could and stores it in the register MODAL instead of V. Then at state VP/V, have satisfies the CAT V arc, so the loop is taken, making have the main verb. Since been also satisfies the CAT V arc, the loop is taken again, and with have in V and been marked as a past participle, PERFECT is added to the ASPECT register and be is placed in the V register, and we re-enter state VP/V with going as the current word. Again, the CAT V arc is permitted, go is made the main verb and PROGRESSIVE is added to ASPECT. Finally we make the jump to VP/HEAD, having identified, as before, the main verb and the subject. The rest of the analysis resembles that of the simple intransitive (9), and the deep structure is similar except that the node AUX has been expanded to (11):

(11)



3. Passives

It was pointed out earlier that passive sentences (12) have the same meanings as their corresponding actives (7). We now show how the grammar maps them into the same deep structure.

(12) Information is needed by me.

The same sequence of transitions is taken for the passive as for the active, up to state VP/V, although the constituents identified and saved in registers differ. Upon entering state VP/V, the register SUBJ contained I and V contained need for the active, while for the passive, SUBJ holds information and be is the main verb as in (13):

(13) Information is available.

At VP/V the analyses diverge. For the active the current word is information, ruling out the CAT V loop, so the jump arc is taken to VP/HEAD, where the object is picked up. For the passive, the current word is needed, the past participle of need. In this case, the CAT V arc is allowed, the subject information is placed on the hold list by the conditional action, the indefinite noun-phrase something is placed in SUBJ, and be is replaced by need in V. At this point, we have identified the main verb, we have partially undone our previous assignment of information as the subject, and the current input word is by. AGFLAG has been set to indicate the possibility that the real subject is in a by-phrase later on. We now make the jump to VP/HEAD, but the push for the object noun-phrase fails with by. Instead, the VIR NP arc removes information from the hold list and places it in the object register. None of the arcs at VP/NP can deal with by, so we jump to VP/VP, where we take the WRD BY transition to VP/AGT, since AGFLAG is set. Here we push for a noun-phrase, find me, and override the indefinite subject something that we set up at VP/V. We continue along the basic analysis path and pop a structure at S/VP identical to that for the active. If the by-phrase had not been found in the sentence, the subject at this point would still be the indefinite something, which agrees with our intuitions about the meaning of passivized sentences with missing agents.

4. Questions

In English, questions introduce a number of variations in the usual subject-verb-object sentence forms handled by the basic strategy. The states emanating from S/Q are designed to cover these possibilities. We recognize three major types of questions, yes-no (14a), question-pronouns and question-adverbs (14b), and question-determiners (14c).

(14) a. Is the document available?
b. What is available?
c. Which document is available?

A yes-no question is characterized by the fact that a modal or auxiliary verb occurs at the beginning of the sentence, before the subject. The predicate QSTART at state S/ precludes any other pre-subject verb, so the jump arc at S/Q brings us to S/NP with only the register TYPE set and with be, have or a modal as the current word. We pick up the verb in the ordinary way, and arrive at S/AUX with a verb but no subject. Hence, we transfer to S/NO-SUBJ where, for (14a), the PUSH NP/ arc is successful, returning the document. This agrees with the person-number code of the verb, and so becomes the subject. From state VP/V the analysis is identical to the corresponding declarative, and the structures are identical except that the question structure has a type node "Q" instead of "DCL".

A question-pronoun or adverb is a WH-word that can stand by itself as the object of interrogation, for example, who, what, when, where, why, and how. The root forms in the dictionary for these items are complete NP or ADV structures, and the CAT QWORD arc makes a copy of this structure (so that other parts of the grammar do not do permanent damage to the dictionary). Most of the pronouns can serve as either the subject or an object of the sentence; these are saved in the register WHQ until further information determines whether they are to be moved into SUBJ or held for the post-verb modifier arcs. The question-adverbs, when, where, and how, and the pronoun whom cannot serve as the subject, so they are held immediately, to be picked up later by VTR arcs. In any case, we enter state S/NP looking at the first verb. If there is a potential subject in WHQ and the first verb is not an auxiliary or modal, then the WHQ word must be the subject, as in

(15), so we rearrange the registers.

(15) Who wants the information?

If the verb is a modal or auxiliary, then the WHQ word is still a possible object, as in (16), so we postpone a decision and enter

(16) What is the computer typing?

S/AUX without a subject. Hence we transfer to S/NO-SUBJ, where for sentences such as (16), we push and recover the full noun-phrase subject. This means that WHQ contains an object, so we add it to the hold-list. For sentences such as (14b), where there is no noun-phrase in this position, we know at last that the WHQ must be the subject, and the registers are rearranged on the jump arc. From VP/V the analysis for QWORD questions follows the basic strategy, except that VIR NP or VIR ADV arcs are taken if the QWORD structure was held.

Finally, questions with question-determiners such as which, what, and howmany (a compound) involve a push from state S/QDET for a full noun-phrase structure with a question-determiner. Since the NP/ network does not recognize WH-words at the beginning of noun-phrases, the determiner must be picked up at the S-level and sent down into the DET register. When the noun-phrase is returned, it is placed in WHQ, and the analysis continues along the lines of the QWORD questions.

5. Existential there.

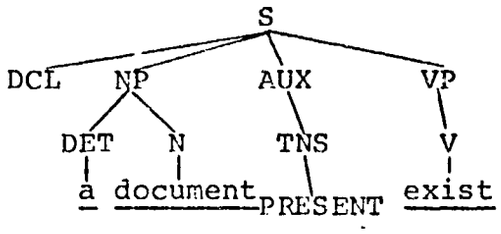
Sentences in which a form of the verb be (or exist) occurs often have counterparts in which the subject is replaced by the word there and the real subject occurs after the be as in (17):

(17) There is a document.

If be is in fact the main verb, the sentence is interpreted as asserting the existence of its real subject. For sentences of

this type, the WRD THERE arc is taken at state S/DCL, setting the register THERE but leaving SUBJ empty. We are still allowed to jump form S/AUX to VP/V, but we cannot go on to VP/HEAD without the subject. Thus, we push for a noun-phrase on the second arc from VP/V, having seen a there-be combination. The noun-phrase a document is returned, and since it agrees with the verb's person-number code, it becomes the subject. The rest of the analysis is ordinary, except that when we finally do jump to VP/HEAD, if the main verb is still be, we convert it to exist. Thus the structure for (17) is (18):

(18)



Notice that this strategy allows the real subject to occur at any position in the string of auxiliary verbs following there, as long as the immediately preceding verb is be or exist. Thus the sentences in (19) can be parse properly:

(19) a. There could be a document.
b. There could have been a document
c. There could have been a document telling about ...

An existential there can also occur in the subject position of a question; hence the WRD THERE arc at state S/NO-SUBJ. If there is a WHQ in this situation, it becomes the subject. Otherwise, the subject is found with the push arc from VP/V, as above. If the WHQ resulted from a QDET question, the DO arc at state S/THERE allows for the resumption of an extraposed noun modifier, as in (20):

(20) How many men were there who wanted the document?

6. Do-support.

In English the verb do can occur as a main verb (21a), as a modal verb with the connotation of emphasis (21b), or as an auxiliary verb in questions and negations with no apparent meaning (21c-d).

(21) a. They did it.
b. The document does contain the information.
c. Did they want it?
d. The document does not contain the information.

The cases exemplified in (21c-d) correspond to the general rules that subjects and verbs can be inverted only if the first verb is an auxiliary; if it is a regular main verb, do is inserted. Similarly a modal or auxiliary must precede the sentential negation operator, and do is inserted if there is no other possibility. In transformational theory, the process of inserting do is called do-support.

The strategy for interpreting do in the DDC grammar is as follows: At state S/NP, where the first verb is picked up, do is placed in the MODAL register, since it satisfies the predicate MODAL. If another verb is not found (as in (21a)), then an action on the jump arc to VP/HEAD moves do from MODAL to V, making it the main verb.

In sentences where the subject and verb have been inverted, the subject is sought at state S/NO-SUBJ. If the PUSH NP/ arc is successful and if do is in MODAL, it is deleted and does not appear in the final parse. Likewise, if the CAT NEG arc is taken at state S/AUX, a do in MODAL is again removed. Thus semantically empty occurrences of do are correctly eliminated, while emphatic and main-verb do's are preserved.

7. Imperatives

The strategy for imperatives is basically simple. An untensed (infinitive) verb beginning a sentence, optionally preceded by please, marks the sentence as a command. Imperatives usually have no overt subject and no modal or auxiliary verbs, so the arc from S/IMP sets up the understood subject you and the tense-indicator PRESENT, and terminates at VP/HEAD where post-verbal constituents are analyzed in the normal way.

8. Objects and complements.

We have already seen in the basic strategy, how simple transitive and intransitive sentences are analyzed. Syntactic features on verbs can require other types of predicate complement structures; the set of paths leading from VP/HEAD to VP/VP allow for the various possibilities, and the annotated listing of the grammar should be consulted to determine precisely how a given sentence will be analyzed. Here we discuss a few common predicate complement forms.

The CAT ADJ arc (arc 4) from VP/HEAD allows the post-verb adjective required by copula verbs such as be, become, appear:

(22) The document is important.

The adjective is placed in the OBJ register and occupies the position in the parse normally taken by the object noun-phrase structure.

The arcs which push to the COMP/ network permit a variety of complement sentence structures. Some verbs can have as their direct object a complete sentence, often preceded by the complementizer that:

(23) I believe that the document is important

Other verbs can have complements beginning with the complementizers for or to, (24) and arc 7 is taken for these constructions.

(27) a. The document seems to be important.

b. We arranged for the document to be sent.

Indirect-direct object combinations are handled by a sequence of arcs from VP/HEAD to VP/VP. In the simplest case, the verb is followed by two noun phrases:

(28) Give me the information.

The first noun phrase is picked up by the PUSH NP/ arc at VP/HEAD, and our initial guess is that it is the direct object, along the lines of (29). When we find the second noun-phrase on arc 4 from

(29) Give the information to me.

VP/NP, we rearrange the registers, making the previous object the object of a dative prepositional phrase, as in (29), and making the second noun-phrase the direct object. The superficial differences between (28) and (29) are thus removed. The grammar allows for various combinations of noun-phrases and sentential complements in the direct and indirect object positions, but we shall not discuss the details here.

9. Verb modifiers

The grammar allows for two types of optional verb modifiers--adverbs and prepositional phrases. These usually occur after the predicate complement structures have been analyzed, and the loops at state VP/VP pick them up. They are added to the list of modifiers in VMODS. Adverbs and prepositional phrases can also occur at the beginning of the sentence, before the normal subject-verb constituents, and the loops at state S/ permit this possibility. Finally, adverbs can occur at other places in the sentence, most often within the sequence of auxiliary and modal verbs; thus,

the CAT ADV loop at state VP/V. Adverbs can sometimes occur in other positions, but the grammar currently will not recognize them.

There is a special arc leaving S/ to deal with negative adverbs such as hardly and barely. When these adverbs occur at the beginning of a sentence (30), the subject and verb are often inverted as in a question. Instead of looking for the subject in

(30) Barely was there enough information

the normal declarative position, the negative adverb arc leads directly to S/NP where the first verb is picked up. The rest of the analysis resembles that of a yes-no question, except that the type is DCL instead of Q.

Finally, in WH-questions where the question word is an adverb (when, how), the adverb is held and picked up on the VIR ADV arc at state S/VP, where it is added to VMODS.

3.2.2 The Noun-Phrase Level

The second major component of the grammar is the noun-phrase level (with state names beginning with NP/). It is entered by pushes from the S-level and prepositional-phrase (PP) states, and it also has recursive calls to itself. We now describe some of the strategies used in the analysis of noun-phrases.

1. The basic strategy

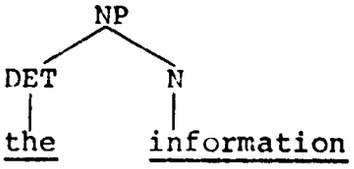
Consider the simple noun-phrase:

(31) the information

At state NP/ the determiner- article the permits the CAT DET transition to state NP/ART. From there a number of arcs permitting optional constituents are by-passed by a series of jumps to state

NP/DET. The CAT N arc picks up the noun ing us to NP/N. We then jump to NP/HEAD and fi which we pop the completed noun-phrase s

(32)



The significant milestones in the analy e are thus as follows: at NP/DET, the series tuents (a simple article in (31)) has been anal iate structure has been built (by the functio d in the register DET. At NP/N a potential h se has been found, while at NP/HEAD the ult deter- mined. Finally, at NP/NP, the complete recog- nized. As at the S-level, more complica recov- ered by variations and elaborations of t ath.

2. Determiner structures

In English it is possible to o before the head of the noun-phrase. For exampl proper nouns, and plural forms of common equire preceding articles, and the jump arc fro rovided for such instances. On the other hand, ore than just an article in the determiner s the analysis of Stockwell et al. (1968), we d quantifiers (with accompanying partitive st- article" structure of determiners. Ordi. sition of the object denoted by the noun-phrase jects (e.g. first, last, next) or along some d llow- est). The constraint is that ordinals p d

other prenominal modifiers, so that (33a) is acceptable but (33b) is not:

(33) a. the next five boys
b. *the five next boys

Since superlative adjectives often serve as ordinals, two arcs (CAT ORD and CAT ADJ) pick up ordinals between NP/ART and NP/ORD and save them in the register POSTART. At NP/ORD, an ordinal, if present, as been recognized.

Following an ordinal, a quantifier is allowed as in (33a). The grammar of quantifiers is fairly complicated, and a separate level (QUANT/) is provided for their analysis. This level is called by the first arc leaving NP/ORD. The QUANT/ states recognize simple cardinals, measure words, and some comparative constructions (more than, less than); this is an area of the grammar that needs further expansion. If found, the quantifier is added to the ordinal structure in POSTART. If a post-article has been identified, a partitive can follow which indicates the set from which the particular object was drawn (34).

(34) a. five of the boys
b. the last of the customers

The partitive is usually introduced by of, and the first arc leaving NP/QUANT looks for an of prepositional phrase. If one is found, the head of the noun-phrase becomes the dummy element ONES, and the partitive phrase becomes the first post-nominal modifier. With the head firmly decided, we transfer to NP/HEAD to look for other modifiers. For certain quantifiers (e.g. all, both) the of can be missing; the second arc at NP/QUANT takes care of this case. Finally, the partitive structure can sometimes be fronted to the beginning of the sentence (35) where a loop at S/ puts it on the hold list for a VIR PP arc to find. If there is no POSTART or if no parti-

(35) Of the documents, how many are about ...

tive is found, the jump to NP/DET is taken, and DETBUILD puts the contents of the DET and POSTART registers into the final determiner structure.

3. Pre-nominal modifiers

After the determiner, a sequence of modifiers can occur before a potential head is found. These may include adjectives, participial forms of verbs, and adverb-adjective phrases. Arcs 1, 3, and 4 at state NP/DET, together with the arcs at NP/ADV, pick up these constituents, saving them in the register ADJS. Examples of noun-phrases with these modifiers are given in (36):

(36) a. the important information
b. a classified document
c. an intriguing fact
d. a very large envelope

4. Other potential heads.

In the basic strategy, the head of the noun-phrase is a noun, picked up on the CAT N arc from NP/DET to NP/N. Two other arcs parallel the CAT N arc, permitting the head of the noun phrase to be a proper noun (37a) (the PUSH NPR/ arc) or a gerund (37b) (arc 5).

(37) a. John
b. the processing of information

In any case, the three arcs place the potential head (embedded in a structure indicating its type) in the register N.

5. Noun-noun modification

It is very often the case that the first potential head in a noun-phrase is not the real head. For example, nouns and proper nouns can be modifiers on other heads (38):

(38) a. acquisition costs
b. data processing
c. United States government

When a potential head is first encountered at state NP/DET, we make the tentative assumption that it is, in fact, the head of the noun-phrase. At NP/N, another potential head implies that the previous head is actually a modifier on the new head; the loops at state NP/N (arcs 4, 5, 7, and 9) pick up the new head and add the old one to the list of modifiers in ADJS. This process can be repeated several times, as in (39a). The CAT ADJ arc leading back to NP/DET means that the series of potential-head-modifiers can have regular adjectives interspersed (39):

(39) a. United States aircraft acquisition costs
b. United States naval vessels

A possessive marker ('s) is also allowed after a potential head has been found, and the CAT POSS arc at NP/N picks it up, converts the previous head to a modifier, and returns to NP/DET. A gerund at this point might confirm that the noun-phrase is really a POSS-ING complement as in (40), and arc 6 at NP/DET pushes into

(40) John's winning of the race

the S/ level to look for this structure.

6. Relative clauses

The jump to NP/HEAD is taken only when the head of the noun phrase has been definitely determined. At this point a variety of post-nominal modifiers is permitted. If we are looking at a

relative pronoun (41a) of a preposition followed by a relative pronoun (41b), we know that there is a relative clause, and we push to R/ to parse it.

(41) a. the documents which we have heard about
b. the documents about which we have heard

Essentially, a relative clause is a sentence with a missing noun-phrase, with the head of the noun-phrase that intuitively should fill the empty slot being the same as the head of the noun-phrase in which the relative clause occurs. Thus we send down a copy of the noun-phrase to be used in the analysis of the relative clause.

The states R/ and R/WH pick up the relative pronoun and decide whether the copy of the noun-phrase sent down (in WH) can be the subject of the relative clause. If so, the noun-phrase is placed in SUBJ. Otherwise, it is held for later use. We then enter the S/ network at S/NP to complete the analysis. The path through R/PREP handles the preposition-relative pronoun case.

The relative pronoun can also be left out of relative clauses, and arcs 4 and 6, which push to R/NIL, handle such "reduced" relative clauses. The reduced relative can begin with a noun-phrase (42a), a participial verb (42b), or an adjective (42c). A noun-phrase is still sent down to WH, and its ultimate

(42) a. the information the document contained
b. the information contained in the document
c. the information available

destination is decided at R/NIL. Notice that there are two arcs that push to R/NIL: the arc usually taken is arc 6, which follows the jump to NP/NP. This means that we will not look for reduced relatives unless we have failed on other paths; this strategy improves the efficiency of the grammar, since R/NIL can lead to long blind alleys. However, this also implies that a reduced

relative on a noun-phrase in a prepositional-phrase within another noun-phrase would be attached to the first noun-phrase in the first parse. Thus in (43) the reduced relative is first a modifier of man instead of the park:

(43) the man in the park the girl frequented

The TST R/NIL arc (arc 4), which calls the SUSPEND mechanism, is included to provide the correct analysis in these cases. It can only be taken if a prepositional-phrase (in the park) has been found in this noun-phrase.

Whenever we find a relative clause, we move to state NP/R, where additional full relatives are allowed until we finally jump to NP/NP.

7. Other post-nominal modifiers

The grammar handles other types of post-nominal modifiers, including prepositional phrases, sentential complements, and parenthetic comments. The PUSH PP/ loop recognizes a sequence of prepositional-phrases and places them, along with the relative clauses, in the register NMODS. The register PPFLAG is set, which enables the TST R/NIL arc and prohibits the normal PUSH R/NIL.

The PUSH FOR/NP loop handles to-complements, such as (44),

(44) the way to do it

which can appear on a wide variety of nouns. The arcs which push to the COMPL/ network can only be taken for certain head nouns, those which take a that-complement as in (45) (e.g. fact, statement, claim). Notice that that can also introduce a relative clause; the difference between a relative clause and a that-complement is that in the relative clause there is empty noun-phrase slot to be

(45) the fact that the documents are not available

filled. The that-complements are complete sentences.

Parenthetic comments are handled in a very special way. Since parentheses in LISP surround a list, and since the basic parsing procedure is set up to operate on lists, we call the parser recursively to analyze the parenthetic comments. The function PARSEPARENS executes the call to the parser and sets up the initial state of the grammar to be state PAREN/. Currently, the only type of parenthetic comment is a noun-phrase, as in (46). The comment

(46) operations research (linear programming)

is interpreted as a non-restrictive modifier on the noun-phrase and is added to the register NR.

Other arcs leaving state NP/NP allow for such constructions as a colon following a noun-phrase and a comma either indicating the beginning of a conjoined sequence of noun-phrases, or introducing a transitive adverb (especially, particularly) and its following noun-phrase. The annotated listing describes these arcs in detail.

Chapter 4

SEMANTIC INTERPRETATION STRATEGIES

4.1 Motivation

In Section 2.3, we presented an outline of the operation of the semantic interpretation component. In this Chapter, we will discuss the particular semantic strategies embodied in the semantic rules and the way in which they extract Boolean topic descriptions from the syntactic structures being interpreted.

The semantic rules used in the system fall into two classes. One class deals with grammatical constructions which have real meaning to the system (determiners, verbal constructions such as "give me", "I need", etc., and noun constructions such as "references on", "reports by", "bibliography of", etc.). The other class deals with constructions whose meaning is not apparent to the system, but which are instead to be interpreted as subject indicators or other restrictions on the documents to be retrieved. We have called this latter class of rules topicrules. We will discuss first the semantic rules of the first type, whose job it is to isolate that part of the syntax tree to which the list of topicrules is to be applied. These rules take care of the ordinary linguistic "amenities" that may occur in a request but which are not directly part of the description of the subject matter of the documents of interest. In a high precision retrieval context, however, they could be an extremely effective means of communicating exactly what the user wants.

4.2 The General Semantic Framework

We have chosen to interpret expressions such as "give me information on ...", "I need a bibliography of references on ...", etc. as having meaning to the system in terms of retrieval routines which will be called to retrieve the appropriate documents. We have done this as opposed to the alternative of simply ignoring such constructions in order to make the system upgradable to other types of retrieval operations which might be incorporated into the system in the future. (Examples might be "How many documents are there which were written after 1950 on the topic of aviation fuels.", "Give me the three most recent Air Force technical reports dealing with short-take-off-and-landing aircraft.", etc.) In such cases, the user could use the determiner structure of his sentences to give very precise and natural specifications of exactly how many references he wants to see, whether he is primarily interested in just knowing the number of references on a topic, etc. Such a system could also be extended to answer questions which are of a more global nature and deal with the structure of the indexing system and not with the documents directly. (Examples could be: "What is a more general term for thrust vectoring?", "What types of aircraft have been used as index terms?", etc.) Thus we have attempted to formulate a set of semantic rules for the operations which are involved in the current DDC application, but we have provided interpretations in a framework which is general and could eventually encompass such additional features as we have just discussed.

4.3 Semantic Representation

The format which we have chosen for the representation of semantics in the system is essentially the same as that described in Woods (1967, 1968), and is based on the quantification of propositions and commands by quantifiers of the form

(FOR QUANT X / CLASS : P(X) ; Q(X)) where QUANT is a quantifier (EACH, EVERY, SOME, etc.), X is the variable of quantification, CLASS is the class of objects over which the variable is to range, P(X) is a restriction on this range (i.e. the only objects in CLASS which are of interest are those for which P(X) is true), and Q(X) is the proposition or command being quantified. In the limited scope of the current project, we are concerned with requests which consist of a single quantification over the class of documents in the system which meet some Boolean topic specification, and the command being quantified is the function PRINTOUT which would print out the answer. Moreover, as we discussed in Chapter 1, for the sake of the current project, the system is not actually connected to the DDC data base, and so the actual documents are not retrieved and printed out by the function PRINTOUT in the prototype system. Instead, the procedure is short-circuited at this point, and the Boolean request is printed out to the teletype. Thus, we have implemented the system with the ultimate goal in mind, and adapted it to the limitations of the current experimental environment, rather than building features of that artificial environment into the system in any essential way.

4.4 Interpreting Sentence Nodes

The interpretation of a sentence node occurs in two phases distinguished by the use of different values for TYPEFLAG. The first phase, with TYPEFLAG NIL determines whether there are any governing operators or commands such as NOT, TEST, etc., which govern the sentence. It is essentially a preprocessing phase prior to the actual examination of the sentence proper and consists in matching rules from the global list PRERULES (which is independent of the particular verb which governs the sentence). Of these rules, S:AND and S:OR interpret conjoined sentences; S:DCL deals with the interpretation of declarative sentences;

S:IMP interprets imperative sentences; and S:WHQ and S:YES/NO deal with questions (the former with questions containing question words such as "which" or "what" and the latter with simple yes/no questions). S:NPU deals with noun-phrase utterances (i.e. sentences which consist only of a single noun-phrase with no verb). Other prerules interpret negative sentences and different syntactic formats.

All of the PRERULES (with the exception of S:NPU and a few others) specify the subsequent interpretation of the same node with the TYPEFLAG SRULES (the exceptions call for the interpretation of specific lower nodes). This second call for the interpretation of the node begins the second phase of processing. In this case, the rules to be tried are taken from the property list (i.e. dictionary entry) of the head of the sentence (i.e. the verb) under the property SRULES. Alternatively, if the verb does not itself have any SRULES, but has as one of its semantic markers a word which has SRULES on its property list, then the rules to be matched will be taken from the list associated with the marker. For example, the rule S:GIVE which interprets sentences of the form "give me information on ..." is used for the interpretation of many words which are synonyms of "give" in this context. This is indicated by putting the semantic marker GIVE in their dictionary entries, and therefore enabling them to use the SRULES from the dictionary entry for the word "give".

4.5 Interpreting noun-phrases

The semantic rules which interpret sentences generally require the interpretation of one or more noun phrases as subconstituents of their interpretation. The rule S:NPU for example, requires only the interpetation of its single constituent noun phrase. Like the interpretation of sentences, the interpretation of noun-phrases occurs in several phases. The first of

these, with NIL TYPEFLAG interprets the determiner structure of the noun phrase to determine what type of quantifier is to govern it. This phase consists of matching rules from the global list DRULES. These rules examine the determiner and number of the noun phrase and assign a basic quantifier structure. They also call for the interpretations of the same node in two different modes--NRULES and RRULES--for the other two phases. NRULES and RRULES both are taken from the property list of the head of the noun-phrase (i.e. the noun) or from the property lists of words which occur in the list of markers for the head noun. NRULES interpret the noun of the phrase and any arguments which it may require (i.e. if it is a function); RRULES interpret any further restrictive modifiers which may occur in the noun phrase. Modifiers which do not match any RRULES are ignored, and relative clauses are handled by a special mechanism which tags the relative pronoun of the relative clause with the variable of quantification and calls for its interpretation as a sentence.

4.6 Interpreting Topics

The above description of the interpretation of noun phrases applies only to noun phrases which have direct and understandable meaning to the system (e.g. "documents on ..." "technical reports", etc.) Other noun phrases consist of topic descriptions and are treated in an entirely different manner by the system. In the latter case, a list TOPICRULES specifies a global list of rules for translating syntax trees into Boolean combinations of key phrases. These rules are grouped on topic rules into AND and OR groups in the way in which any resulting matches are to be combined. Each topic rule corresponds to a particular type of key phrase which may be present in a syntactic construction, and specifies in its left-hand side the proper context and structure for the extraction of that key phrase.

The list of TOPICRULES is used to interpret a noun phrase instead of the usual sequence of DRULES, NRULES, and RRULES whenever the call to interpret the noun phrase is made with TYPEFLAG TOPIC instead of TYPEFLAG NIL. Semantic rules of the ordinary type are used to invoke this special type of interpretation whenever they locate a context which is definitely a topic. For example, the rule R:DOC-ON interprets restrictions on a noun which is semantically marked DOCUMENT that begin with the preposition "on". It interprets such constructions as "data on X" by calling for the interpretation of X with TYPEFLAG TOPIC and constructing an instance of the ABOUT predicate indicating that the documents in questions are about the topic X.

In many cases in the sample requests which we received, the user omits specific reference to documents or reports and proceeds directly to describe the topic he is interested in. Thus, either of the requests "Please run a search for documents on polar bears." and "Please run a search on polar bears." are likely to occur. For this reason we have implemented a special TYPEFLAG REFS? which is used to call for the interpretation of a noun phrase in a context where it may either explicitly mention documents or simply describe a topic. When the interpreter is called with this TYPEFLAG, the function RULES returns the list (REFRULE? NIL REFRULE), where the NIL indicates that the second rule is to be tried only if the first one fails. The rule REFRULE? asks whether the node can be interpreted in the normal manner (i.e. has a head which is meaningful to the system (currently the system only knows about documents) by calling INTERP in the normal mode. If so, then this normal interpretation is the one assigned to the node. If not, then the rule REFRULE is applied which generates a quantifier over documents qualified by an instance of the ABOUT predicate using the interpretation of the current node as a topic. That is, if a node X is not interpretable in the normal mode, then it is interpreted as "documents on X" when the TYPEFLAG is REFS?.

4.7 Padding Adjectives

In many cases, adjectives used in the user's request convey no real content information. An example is the word "comprehensive" in the phrase "comprehensive bibliography". In a future system, such information could be used to specify the amount of effort to be devoted to the search or the amount of effort to be devoted to the search or the amount of material which the user wishes to receive (as opposed to "a representative bibliography", for example). However, in the current syste, since there is no facility in the target DDC retrieval language for such information to be used, such modifiers are ignored.

We have defined two classes of nouns and adjectives, "document" and "padding", which occurring alone or together are ignored. (Class membership is indicated by the semantic markers PADDING and/or DOCUMENT.) "Comprehensive" is marked "padding" and "bibliography" is marked "document". The phrase "comprehensive bibliography" is therefore ignored. We can compare this to request 35476 in which the phrase "unclassified documentation" occurs. "Documentation is marked "document" and alone, would be ignored. However, "unclassified" is a content word,* so the phrase "unclassified documentation" is a valid keyphrase. The strategy is:

(1) alone, a "padding" or "document" word is ignored

(2) if modified by another "padding" or "document" word, the whole noun phrase is ignored.

(3) if modifying or modified by a content word, the noun phrase is taken as a keyphrase.

*However, "unclassified" is not a subject indicative keyphrase, but rather specifies an attribute of the reports themselves. In a full blown system with a complete retrieval component, "unclassified" would be handled in a different way. It is included as a keyphrase at the moment because it is important information which should not be discarded.

The set of words marked either "padding" or "document" is not fixed. A word may be so marked by using the LISP function ADDPROP (e.g. ADDPROP(PAMPHLET MARKERS DOCUMENT), or ADDPROP(THINGAMOBOB MARKERS PADDING).) A word may be removed from either class by using the function REMPROP (if there is only one thing in the list of MARKERS) or editing the property list of the word in question and removing "padding" or "document" from the list of MARKERS.

The words currently marked "padding" are:

appropriate,aspect,available,broad,complete,
comprehensive,following,ones,printed,printout,
recent,something,specific,technical,type.

The words currently marked "document" are:

article,bibliography,citation,data,directive,
document,documentation,information,list,listing,
literature,material,paper,publication,reference,
report,research,study,survey,thing,title.

4.8 Two Examples of Interpretation

In this section, we trace the translation of two syntactically dissimilar requests into the intermediate semantic representation used in the DDC Pre-Processor System. The two requests to be followed, taken from the DDC sample set, are:

36269 A COMPREHENSIVE BIBLIOGRAPHY ON POLAR BEARS

36807 WE ARE INTERESTED IN ANY REPORTS WHICH HAVE INFORMATION CONCERNING ELECTRONIC TECHNIQUES FOR THE SUPPRESSION OF AUDIELE NOISE.

The intermediate semantic interpretations that are produced by the above two requests are:

36269 (FOR GEN X4 / DOCUMENT : (ABOUT X4 (OR (POLAR BEAR) (AND (POLAR)(BEAR)))); (PRINTOUT X4))

```
36807  (FOR SOME X13 / DOCUMENT : (ABOUT X13 (AND (OR
       (ELECTRONIC TECHNIQUE)(AND (ELECTRONIC)(TECHNIQUE)))
       (OR (SUPPRESSION OF AUDIBLE NOISE)(AND (SUPPRESSION)
       (OR (AUDIBLE NOISE)(AND (AUDIBLE)(NOISE)))))))) :
       (PRINTOUT X13))
```

The input to the semantic interpreter is a syntactic tree description of the request, produced by the parser described earlier. This, it examines from top to bottom, forming the interpretation of the whole request from the interpretations of its parts. At the lowest level, individual phrases and words are interpreted as keyphrases, what the documents to be retrieved are ABOUT. At higher levels, the keyphrases are combined appropriately, the proper quantifiers added, an intermediate semantic interpretation produced and alternative ones, if any, considered. Only demands for documents are interpretable. If the input cannot be interpreted as a request for documents, it is not interpretable, and control is returned to the parser for another analysis. If a subsequent parsing is capable of interpretation, things procede as above. Otherwise, when the parser exhausts all possible analyses (or an arbitrary limit - currently 5 attempts to reparse), the input is rejected.

4.8.1 Example One

```
36369  A COMPREHENSIVE BIBLIOGRAPHY ON POLAR BEARS
       S  NPU
          NP  DET  NIL
              ADJ  COMPREHENSIVE
              N BIBLIOGRAPHY
              NU SG
              PP  PREP  ON
                  NP  DET  NIL
                      ADJ  POLAR
                      N  BEAR
                      NU  PL
```

The function RULES directs the interpretation. It first attempts an interpretation of the whole sentence, S. Using the list of rules, PRERULES, it finds that the sentence is a noun phrase utterance (NPU) and should receive the interpretation attached to its main noun phrase (NP). At this point, the right-hand side of the rule (PRED (PRINTOUT (# 1 1 REFS?))) indicates that the interpretation is a predicate (which may later be quantified) governing the command PRINTOUT. It indicates that the things to be printed out are to be determined by interpreting the noun-phrase (# 1 1) with the typeflag REFS? (i.e. the noun phrase may be either a topic description or a noun phrase whose head is a class of documents). The function PRED will be executed after the substitution of the interpretation on the noun-phrase has been made in the right-hand side, and it will grab any quantifiers which have been produced by the constituent interpretations.

The interpreter now begins the interpretation of the noun phrase using the two rules REFRULE? and REFRULE (determined by the TYPEFLAG REFS?). The first attempts the interpretation in the normal mode beginning with the global list DRULES. Since the noun phrase does have an interpretable head ("bibliography") this interpretation will succeed, and the rule REFRULE will never be tried.

As we mentioned, the DRULES are used to interpret the determiner and number of the noun phrase and determine the type of quantifier to be produced. This includes the case of no determiner (determiner NIL). In this case, the special generic quantifier GEN is produced, and placed in a buffer string for the function PRED to grab. A variable of quantification (X4) is assigned to the noun phrase, and the interpreter is called with the TYPEFLAG NRULES to interpret the noun.

In the dictionary, the word "bibliography" has no NRULES listed, but it is semantically marked as DOCUMENT, and the word DOCUMENT has NRULES on its property list. Hence this list of NRULES is used for the interpretation. The rule N:DOCUMENT (the only real NRULE in the current system) specifies that the class of quantification is DOCUMENT, and the interpretation of the restrictive modifiers with TYPEFLAG RRULES is called for. The list of RRULES is also taken from the dictionary entry for DOCUMENT, and the rule R:DOC-ON specifies an instance of the predicate ABOUT is a restriction and directs the interpretation to the noun phrase "polar bears" with TYPEFLAG TOPIC.

Many of the TOPICRULES calls for the interpretation of embedded constituents also with TYPEFLAG TOPIC; thus TOPICRULES, like the other semantic interpretation rules, are fundamentally recursive. The TOPICRULES applicable in this example are TOPIC\TERM, TOPIC\ADJ, and TOPIC\N. (A full description of the TOPICRULES is given in Appendix C.) TOPIC\TERM is the rule which produces a keyphrase for the entire construction if the length is sufficiently short (not more than four words); in this case it yields "polar bears". TOPIC\ADJ selects the adjective of a noun phrase as a possible keyphrase if it is not a padding adjective; in this case it yields "polar". Similarly, TOPIC\N selects the noun of a noun phrase as a keyphrase if it is not marked DOCUMENT, and in this case yields "bear" (note the standardization to the singular form--this feature eliminates the possibility of missed matches due to inflectional variation. The plural also could have been chosen as the standard form). TOPICRULES also specifies the correct combination of these keyphrases as a topic - (OR (POLAR BEAR) (AND (POLAR) (BEAR))). The result of these functions is the intermediate semantic interpretation given at the start of this section:

```
(FOR GEN X4 / DOCUMENT : (ABOUT X4 (OR (POLAR BEAR)
(AND (POLAR)(BEAR)))) ; (PRINTOUT X4)).
```

4.8.2 Example Two

36807 WE ARE INTERESTED IN ANY REPORTS WHICH HAVE INFORMATION CONCERNING ELECTRONIC TECHNIQUES FOR THE SUPPRESSION OF AUDIBLE NOISE

```
S  DCL
   NP  PRO  WE
   AUX  TNS  PRESENT
   BP V  BE
      ADJ INTERESTED
      PP  PREP  IN
          NP  DET  ANY
          N  REPORT
          NU  PL
          S  REL
             NP  DET  WHR
                 N  REPORT
                 NU  PL
             AUX  TNS  PRESENT
             VP  V  HAVE
                 NP  DET NIL
                     N  INFORMATION
                     NU SG
                     S  REL
                        NP  DET WHR
                            N  INFORMATION
                            NU SG
                        AUX  TNS  PROGRESSIVE
                        BP V CONCERN
                           NP  DET  NIL
                               ADJ  ELECTRONIC
                               N  TECHNIQUE
                               NU  PL
                               PP  PREP  FOR
                                   NP  DET  THE
                                       N SUPPRESSION
                                       NU SB
                                       PP  PREP  OF
                                           NP DET NIL
                                              ADJ AUDIBLE
                                              N NOISE
                                              NU  SG
```

Using the list of rules, PRERULES, the interpretation function, RULES, finds that the input is a declarative sentence and that its interpretation depends on rules attached to the main verb. RULES is directed to the definition of the main verb BE, to find the property SRULES. This contains a list of rules for interpreting sentences in which the verb is "be". In this example, the verb "be" has many different rules which specify its interpretation in many different situations. For example, documents can BE ABOUT some subject, or, in this case, people can BE INTERESTED IN some subject. Each rule specifies an environment of the verb BE in which the rule is applicable. The rule S:BE-INTERESTED directs RULES to the interpretation of the object of the preposition, IN. At this point, the interpretation is specified by (PRED (PRINTOUT (# 3 2 REFS))), where (#3 2) is a reference to the noun phrase hanging off the preposition IN.

Next, the noun phrase is examined for quantifiers. ANY, like SOME, specifies the interpretation

(QUANT (FOR SOME X / (# 0 NRULES) : (# 0 RRULES) ; DLT)).

From this point, the interpretation proceeds as in the first example. Because several different TOPICRULES are used, we shall continue to follow the steps towards an interpretation.

REPORT, like BIBLIOGRAPHY, is marked DOCUMENT, and the same NRULES and RRULES apply. We are left to consider the noun phrase headed by ANY REPORTS as the source of what the DOCUMENTS are ABOUT. The following topic rules are matched (indentation indicates recursive calls to the interpreter):

TOPIC\TERM - rejected: noun phrase too long
TOPIC\N - rejected: head noun marked DOCUMENT so not a topic
TOPIC\REL - directs RULES to the interpretation of the first embedded S node (a relative clause).

TOPIC\TERM - rejected: noun phrase too long

TOPIC\N - rejected: head noun marked document

TOPIC\V-TRANS - (VP: HAVING INFORMATION). This rule was not operational at the time of our major run, and currently does no semantic checks on the verb and its object to discard non-content phrases as that above. Its improvement will be part of any further work.

TOPIC\S.OBJ - directs RULES to the interpretation of the second embedded S node.

TOPIC\TERM - rejected: noun phrase too long

TOPIC\N - rejected: head noun marked document

TOPIC\V-TRANS - yields (VP: CONCERNING ELECTRONIC TECHNIQUE)

TOPIC\S.OBJ - directs RULES to the interpretation of the noun phrase headed by TECHNIQUE.

TOPIC\TERM - rejected: noun phrase too long

TOPIC\ADJ-N - (ELECTRONIC TECHNIQUE)

TOPIC\ADJ - (ELECTRONIC)

TOPIC\N - (TECHNIQUE)

TOPIC\PP - directs rules to the interpretation of the noun phrase headed by SUPPRESSION.

TOPIC\TERM - (SUPPRESSION OF AUDIBLE NOISE)

TOPIC\N - (SUPPRESSION)

TOPIC\PP - directs RULES to the interpretation of the noun phrase headed by NOISE

TOPIC\TERM - (AUDIBLE NOISE)

TOPIC\ADJ - (AUDIBLE)

TOPIC\N - (NOISE)

These keyphrases are conjoined in a manner specified by TOPICRULES, and the interpretation of this request is that noted at the beginning of section 4.8.

Chapter 5

PERFORMANCE AND EVALUATION

5.1 Performance on the Initial Sample

The DDC English Preprocessor has been run in a series of "batch" passes on the entire sample of 200 English sentences provided by DDC at the beginning of the project. Of this set, 162 or 81% have been successfully parsed and interpreted, thus satisfying the target goal of 80%. However, the quality of the Boolean requests produces is far below that which could be produced by a human analyst, and the system as it now stands would not be adequate for a production system. The reasons for this are several and will be discussed in detail in subsequent sections.

Of the 38 sentences which were not parsed, there are 10 which look like they should have been parsed by the current system. The failure of these sentences is probably due to minor errors in the dictionary coding of some of their words. The English preprocessor is critically sensitive to the dictionary entries and the absence of single feature on a word may prevent the parser from finding a parse of a given sentence. In fact, three of these ten senteces had been parsed earlier in the contract when we were experimenting with the system and testing it on individual sentences (in a mode in which errors in dictionary coding could be corrected dynamically on line). In another of the cases, the failure in the "batch" mode appears to be the result of a misspelling in the sentence. Limitations in time have prevented us from examining the cause of failure of these sentences in more detail.

In addition to the sentences whose failure is apparently due to trivial bugs, there were 11 sentences in the original sample which we judged to be nonsense or syntactially bad. We would not

want to expand the grammar to incprporate such sentences in any case (since the introduction of "bad" grammar rules into the grammar will introduce a potential for "bad" parsings for otherwise acceptable sentences). The remaining 17 sentences which failed to parse either contained constructions which were not included in the grammar or were sufficiently complex that a parsing was not found within the maximum time alloted for parsing (60 seconds).

5.2 Execution Time

Timing the execution of algorithms in a paged time-sharing environment is a difficult task. The elapsed time between the typing of a request and the receipt of an answer (the time which is most directly apparent to the user) is affected by a variety of factors that are variable and depend on factors other than the amount of computer time actually required to do his job. This elapsed time (or "real time") is a function not only of the speed of the machine and the difficulty of the job, but also of the storage capacity of the machine, the efficiency of the time-sharing monitor, and the distribution of other jobs that are running at the same time. Because of its dependence on so many extraneous variables, the real time required by a job is not a reliable figure for comparing algorithms run on different machines or for projecting performance from one machine or operating system to another.

The BBN-LISP system has access to various system clocks and can measure the actual amount of central processor time used by a given task. This timing figure is much less sensitive to the number of other users on the system, and is the only figure which can be reliably projected to another machine or operating system. Even this figure, however, depends on the loading of the system due to the increased frequency of paging and the overhead for being swapped in and out of core. Timing figures should therefore

be taken as providing an order of magnitude for the time required by an algorithm, but are not invariant. Subsequent measurements of the same algorithm on the same problem will not reproduce the same timing result.

In the normal mode of running the English preprocessor system, the amount of time spent in each of the two phases of parsing and semantic interpretation is recorded and printed out with the results. We computed the average parsing and semantic interpretation times for one output file consisting of 81 successfully parsed and interpreted sentences (half of the total number of sentences parsed). The average time spent in parsing was 8.5 seconds, and the average time for semantic interpretation was 22.8 seconds. Thus, the overall processing time for these sentences is on the order of half a minute. These figures are the averages for sentences which succeeded in parsing. We did not compute the average amount of time required to decide that a sentence did not have any parsings, but it is definitely much longer since many of these sentences failed by exceeding the 60 second time limit. If we assume that each sentence which failed to parse took the maximum time of 60 seconds (which is not far from the true value although it is a little high), then the average parse time over 100 sentences (81 of which parsed) would be 18.3 seconds. The average time spent in interpreting the sentences which do not parse is of course 0, so that the average interpretation time over 100 sentences is 18.5, or the average total of parsing and interpretation time is 36.8 seconds (still on the order of half a minute).

An overall improvement in the average execution time could be gained at the expense of failing to get some parsings by lowering the ceiling of 60 seconds on a parsing.

5.3 Major Problems

By far the most important problem that emerges from our experience with the initial sample is the difficulty of mechanically determining the most likely scope for reduced conjunction constructions. We have adopted a compromise solution which gets an acceptable parsing most of the time, but in many cases, there are other parsings which are better, and in other cases the first parsing is simply wrong. There are probably some more purely syntactic heuristics which can be discovered and used to improve the current performance, but we feel that the ultimate solution requires the use of extensive semantic information, some of which may be obtainable from the data base of the retrieval system.

Other major problems which the current project has brought to light or at least more sharply into focus are the difficulties in providing useful feedback to the user, and the importance of a large, well debugged dictionary.

5.3.1 "and" or "or"

One of the interesting things about the use of conjunctions in the samples is every explicit occurrence of the word "and" is ambiguous. That is when a person asks for information on air pollution and water pollution, does he want only documents which deal with both air pollution and water pollution, or does he want documents which deal with either air pollution or water pollution. That is, the user may mean "or" even though he says "and". Indeed, in most cases in the sample sentences, this seems to be the preferred interpretation. That is, when the user says "and", he usually means "or". However, there are a number of requests in the sample set in which "and" is used as a Boolean connective for a request which is already in the form of a Boolean combination of descriptors. In this mode, the "and" means "and".

Since the system cannot tell in general which type of a request it is receiving, it is forced to make its decision uniformly by one of the following strategies:

1. The user says what he means (i.e. "and" means "and").
2. The user always means "or" when he says "and".

The second alternative has the serious drawback of leaving the user no other way to say "and" when he means it, although in terms of numerical occurrences in the sample, it is the correct interpretation more often than not. Therefore, we have adopted a convention that the user says what he means, and that he will be sufficiently disciplined that he will say "or" instead of "and" in those cases where that is what he really means. This should not be too great a problem, since the ultimate application in which this system is viewed as operating is one in which the user will rephrase his request several times as a result of seeing what he gets from the first request. Thus the first request will, in addition to showing him the number of hits for each of the individual phrases as well as those for the request as a whole, show him the interpretation of his conjunctions (thus reminding him that he should have said "or" if that is what he meant) so that in his second request, which he would normally make anyway, he will also be able to use the correct conjunction.

Although we believe that the above solution is the best one in general, there may be users who are sufficiently set in their ways of using English that they want to say "and" without thinking about it and have the system interpret it as "or". Since the change in the system required to produce this behavior instead of the other alternative is a minimal change, we have provided a function (ORFLAG) which will set up the system to operate in this mode instead of the normal one. Thus the user has his option as to how the system will behave with respect to this phenomenon. This is one example of the system's capability for adjusting itself to needs of individual users.

5.3.2 Reduced Conjunction Scoping

In addition to the problem of ambiguity of the word "and" that arises from its use as an implicit "or", there are further problems associated with the use of "and" (and also "or") in reduced conjunction constructions. These are constructions in which some modifier which is common to both components of the underlying conjunction has been "factored out" either to the left or the right and appears only once in the surface sentence. For example, the phrase "recent articles and technical reports dealing with lasers" would most likely be interpreted as the conjunction of the two phrases "recent articles dealing with lasers" and "recent technical reports dealing with lasers." In this interpretation, the common modifier "recent" has been factored out to the left, and the common modifier "dealing with lasers" has been factored out to the right. However, the same phrase also permits interpretation as the conjunction of the two phrases "recent articles" and technical reports dealing with lasers" (completely unreduced), "recent articles" and "recent technical reports dealing with lasers" (only the left modifier has been factored), etc. For human parsing, a variety of syntactic and/or semantic clues are used in determining which of these possibilities are intended by the user (for example, we know that a request for "recent articles" without any topic specified is an unreasonable request from the user (although a perfectly meaningful one) and that "technical reports" is a compound word analogous to "articles"). For the mechanical parser, the same types of information are not usually available. The provision of a natural language parser with fluency in English comparable to a normal English speaker is far more than a year's effort. People spend at least 6 years of their life during childhood working virtually full time to develop their fluency and the process never stops. Moreover, linguists have been studying natural language for centuries, and have still very little understanding of exactly what is going on in such constructions as conjunction reduction. It is not surprising therefore that there is considerable room for improvement in the treatment

of conjunction reduction by the present system. The current grammar has made a considerable advance in the area of conjunction handling over most of the available state of the art (which usually deals only with conjoined complete constituents and leaves conjoined sentence fragments and reduced conjunctions alone), but it still falls far short of the fluency of a human in dealing with such constructions. This is perhaps the major hurdle to be dealt with in the development of a practical natural language querying facility of the type envisioned. The short run solution lies, we feel, in the use of conventions to impose formal restrictions on the ways in which conjunctions can be used (this was not one of the options in the current contract), and the long range solution appears to require the use of extensive semantic information and various heuristics for selecting the most likely scopes for reduced conjunctions. It is likely that access to the data base of the system to determine what key phrases exist in the thesaurus could be used to provide some of this information and therefore greatly improve the selection of good conjunction scopings, but the determination of whether such is in fact the case will have to wait until such access is available.

5.3.3 Commas

One of the problems that causes parsing failures in the current system arises out of the ambiguity and elusive nature of commas in English. Almost all existing natural language parsing grammars either do not permit commas, or else deal with them only in one specific construction (such as A,B, and C). In English, however, and in the DDC sample in particular, commas are used for a number of purposes. In the DDC sample, commas are used in comma conjoined constructions of the form A,B, and C; they are used to introduce post-nominal modifier constructions such as "including ..." and "especially ..."; and they are used to separate examples enclosed in parentheses following a noun phrase. In addition, they are occassionally misused in places where commas would not

normally be permitted in grammatical English, and they are infrequently used to separate adjectives modifying a single noun. We have provided in the grammar for the former three cases, but the latter two (ungrammatical uses and use to separate adjectives in a noun phrase) have not been provided for in the grammar. Thus a certain number of sentences fail to parse because of uses of commas other than those provided for in the grammar. (Note; in the current system, comma conjoined constructions of the form A,B, and C are permitted only for complete noun phrases, and not for adjectives or for prepositional phrases. At least one sentence in the sample failed to parse because of using comma conjoined adjectives in this way, but the number of such instances is not large. These constructions could be included in the grammar without much difficulty, and were omitted only because of limitations of effort available.)

In addition to sentences which failed because of improper use of commas, there were other sentences which failed through failure to use a comma where one was indicated. Thus several sentences with post-nominal modifiers of the form "including A and B" failed to parse because of failure to introduce them with a comma. Again, the facility to accept such constructions without a comma can be added to the grammar with relatively little effort.

In other cases, the system has had problems with the correct interpretation of commas which were correctly used, but potentially ambiguous as far as the system's grammar is concerned. This problem is pronounced in the parsing of reduced comma conjunctions. More work needs to be done in this area, and a rather major effort in the area of parsing of conjunctions remains to be done (even though considerable advances over the general state of the art were made in the current project in order to reach the performance level that we currently have).

5.3.4 Feedback to the User

In the typical instance, the user's first request will not be likely to provide exactly what he wants. (Indeed, in the current DDC operation, the user's first request frequently results in a telephone call from the DDC analyst asking him to clarify what he "really" wants.) Instead, his first request should give him information about whether the terms he used are in the thesaurus or the data base and how many documents are indexed under each term. Also it will display for him the Boolean combination of terms which it has constructed from his request, so that in his second request, he will have feedback concerning both the terms that he used, and the way that he put them together. For example, if he said "and" in his request where he actually meant "or", then in reviewing the Boolean request he would discover that fact and could adjust his wording accordingly. Thus, the performance which will be critical in the ultimate system will not be as much concerned with the success of the first request, but with the types of feedback information provided for rephrasing his request. Since the terms of the present contract call for performance specifications on the former and not the latter, our effort has been devoted largely to meeting the 80% performance on the first request, and much work remains to be done in the area of providing useful feedback and diagnostic comments. We have performed a number of experiments in this area, and the system contains some basic facilities for error diagnostics, but the current facilities for such feedback fall short of that required for a system which could be used by a naive user with no instruction. Indeed, the provision of such feedback has been a perennial problem in computer operating systems in general, and is still an open research problem. The complete solution of the problem in the DDC context may require considerable experimentation and research.

5.4 The DDC Evaluation Period

For a period of two weeks during the month of April, the system was used on-line by the staff of the Defense Documentation Center over a telephone hookup to the BBN time-sharing system. During this period and an additional two days at BBN in May, a trial set of 100 sentences which we had never seen before (and indeed did not see until after the testing period) were processed. This period of evaluation was very educational and highlighted a number of important aspects of the problem.

During the DDC trial period, the English preprocessor parsed 77 of the 100 sentences and translated 76 of them into Boolean strategies, satisfying the target goal of 75%. Again, however, the quality of the resulting search strategies was inferior to that which one would have liked. The DDC staff members who evaluated the resulting Boolean strategies rated only 39 of the strategies as "accurate" --(that is, accurately reflecting the intention of the requestor, or likely to produce a result that will satisfy him). BBN's evaluation of the same sample with slightly different criteria rated an additional 17 requests as "correct" (that is, accurate literal interpretations of syntactically correct parsings of the requests) and another 7 requests as "acceptable" (that is, Boolean strategies which would not be totally misleading and which would provide useful information relevant to the request).

Two problems were highlighted by the DDC trial evaluation in addition to the problems discussed previously. One of these is the importance of having a completely debugged dictionary for every possible word which can be meaningfully used in a request, and the other is the necessity (at least with the present state of the art) for the user to exercise care in phrasing his requests to be sure that he says what he means. In the next two sections we will discuss these two problems in detail.

5.5 Dealing with Unknown Words

Much of the behavior of both the parser and the semantic interpretation system is conditioned by the presence of syntactic and semantic features in the dictionary entries of the words of the request. Thus the existence of these features in the dictionary entries is crucial to the successful processing of a request. In the experimental operation of the prototype system during the trial period, it was inevitable that the system would encounter words that were not yet in its vocabulary, and we spent some effort early in the contract attempting to provide a facility for the on-line addition of English words to the system dictionary. A basic facility which allowed a linguist or a trained lexicographer to make such entries was implemented immediately and has remained in the system as a primary tool for the development of the current dictionary. However, it was clear that the typical user of the system would not possess the linguistic knowledge and the knowledge of the system's formats which were required by this facility.

At one point, we developed dialog facility for entering syntactic information which eliminated the need for a user to know the details of the formats of the system dictionary, but still required an intimate working knowledge of English parts of speech--a topic which unfortunately is covered only briefly for a few weeks in the high-school curriculum and which is exercised not at all in one's daily life. Since almost no one remembers the details of this portion of his education without further training, we could not expect an untrained user to use this facility. Moreover, in addition to requirng detailed linguistic knowledge, this facility proved bulky and cumbersome and was therefore abandoned. A possible alternative would have been for the system to ask questions about the uses of a word which were even more basic than parts-of-speech classifications, but such a facility would have been extremely tedious to use and would have been intolerable to even the most patient user. This approach was not even attempted.

Still another possibility for dealing with unknown words is to attempt to infer their parts of speech from the context of their occurrence and features of their spelling. Such systems have been developed elsewhere and could possibly be used for this application, but this procedure involves an element of guessing, and one would still want such entries to be scrutinized by a trained lexicographer before entry into the system's permanent dictionary. Also, this procedure is fundamentally limited to guessing the syntactic part of speech only, and is not capable of inferring the syntactic and semantic features to associate with the words. Thus a human lexicographer would be required in any case to provide this information.

Our conclusion was that the best way to deal with unknown words for the eventual DDC application was to make them virtually nonexistent. That is to provide the system with a large working vocabulary of general English as well as a complete dictionary entry for every possible term that may lead to the successful retrieval of a document. In this way, the system need only inform the user that a word is unknown and he will know that the word would not have lead to a successful retrieval even if the system were given the necessary syntactic information. The dictionary in this case would be maintained by a trained lexicographer who could afford to spend the necessary time on each new word to construct the proper dictionary entry for all of its uses. New words for consideration by the lesicographer could be provided automatically by both the indexing system and by the retrieval system. The indexing system would inform him of all new words which have been used to index documents, and the retrieval system would provide lists of unknown words which occurred in users' requests.

The running of a successful production system on this basis would require the construction of a comprehensive dictionary by trained lexicographers, an undertaking of sizeable magnitude, but once the initial dictionary was constructed, the task of maintaining it would not be onerous.

Within the level of effort provided for by the current contract, the construction of such a massive dictionary was obviously out of the question. We did construct a 3500 word dictionary which included all of the words in the initial sample as well as a large vocabulary of general English and technical terms. To this we added an additional 500 words and supplementary information for another 500 words that were provided by the DDC staff prior to the trial period. For the additional entries for words that occurred during the trial period and were not already in the dictionary, we were forced to rely on the DDC staff members using the on-line facility that was designed for the system's programmers at BBN and a manual of dictionary formats with which we provided them. Unfortunately, due to the shortness of time and the limited training period for the DDC staff who performed this function, it was inevitable that there would be errors made in this process, and a fair portion of the unacceptable Boolean search strategies during the trial resulted from errors in entering dictionary entries on line. (Some others resulted from residual errors in the 4000 word dictionary.)

5.6 Literal Interpretation of Requests

One of the characteristics of human communication is that a person puts as little effort into phrasing his sentences as he finds necessary to make himself understood. Since human listeners are very adept at understanding what is intended even when a sentence is garbled or misspoken, the effort devoted to phrasing is usually minimal, and the sentences produced in this mode do not always say what they mean. The converse of this observation is also true, in

that a person will put more effort into phrasing his sentences if he finds it necessary to make himself understood.

It would be nice to have a machine which is as adept as another person at understanding what a person meant to say as opposed to what he said. However, such a capability is well beyond anything in the current state of the art in natural language processing or artifical intelligence. It requires an ability to not only interpret the sentence literally (which the current English preprocessor does) but also the judgement to decide that the literal meaning is not reasonable (i.e. the requestor probably didn't mean that). This in turn requires an extensive amount of knowledge about the subject matter of the request, what constitutes a reasonable request, etc. Such a facility could probably be developed, but it would require at least several years of research. Moreover, it is not clear that such a facility would be worth the cost of developing it for the DDC application.

In an on-line environment , being occassionally misunderstood does not cause appreciable difficulty so long as the user is provided with feedback which tells him how his request was interpreted. When he sees that he has been misunderstood, he can rephrase his request accordingly, and as he becomes accustomed to the literal way in which the machine interprets his requests, the number of such misunderstanding will decrease. Thus, we do no consider this aspect of the system a serious problem in the intended DDC environment, although it accounts for a number of the Boolean strategies which were judged inaccurate by the DDC staff.

5.7 Postscript

Subsequent to the trial period itself, the correction of minor bugs in the grammar, dictionary, and semantic rules (mostly in the dictionary) resulted in 18 additional interpretations (6 additional parses and 12 new interpretations for requests which had parsed

previously) which we judged to be "correct" or "acceptable". The overally breakdown of the performance on the trial sentences (including the post-trial processing) is shown in Figure 5-1.

When added to the 39 interpretations judged accurate by DDC staff, the 17 interpretations which we evaluated as correct representations of what the user said, and the 7 interpretations which we considered "acceptable" interpretations, these new interpretations bring the total number of "acceptable" interpretations to 81. Thus, with a complete and carefully debugged dictionary, the current prototype could provide approximately an 80% performance level of marginally acceptable interpretations. We do not feel however, that the quality of these interpretations is sufficient for the implementation of a production system. In Chapter 6, we will discuss more fully where we now stand and the prospects for the future.

OK	N	G	A	XX	XP	XI
48111	48197	48271	48255	48125	48295	48137
48257	48256	48293	48294	48139	59825	48296
48263	48259	48374	48405	48267	59948	48315
48285	48268	48404	59860	48301	59958	48324
48307	48283	59816	59981	59823	59973	59956
48308	48298	59817	60068	59967	59989	60065
48309	48300	59819	60069	59986		
48311	48302	59877			6	6
48323	48389	59878	7	7		
48352	59867	59880				
48382	59868	59942				
48741	59869	59979				
59813	59964	60000				
59815	59980	60047				
59818	59985	60053				
59870	60003	60060				
59875	60057	60062				
59876	60058	17				
59951	18					
59953						
59955						
59957						
59963						
59966						
59974						
59982						
60001						
60005						
60007						
60008						
60011						
60017						
60020						
60021						
60022						
60046						
60051						
60054						
60061						
39						

OK DDC rated strategy OK

N new interpretation (other than that gotten by DDD during test) which is correct or acceptable

G good strategy for what the user said, but DDC rated it bad or didn't say

A acceptable

XX ungrammatical or bad input sentence

XP failure to parse

XI failure in interpretation (may be due to incorrect parse)

Figure 5-1

Performance Evaluation on the Trial Sample

Chapter 6

SUMMARY AND CONCLUSIONS

1. Summary System Description

The DDC English Preprocessor System is a prototype system which was constructed to give the DDC a realistic look at the current state of the art in natural language processing as it would apply to the use of natural English queries in the DDC on-line environment. In actuality, the system is pushing the limits of the current state of the art, and some of the techniques that have been used are at the frontier of current research or slightly beyond. The system makes use of a transition network parser and semantic interpretation procedure developed at Harvard University and at Bolt Beranek and Newman (Woods, 1968, 1969, 1970) and performs a detailed syntactic analysis of the input sentences, determining the parts of speech of the words and the syntactic relationships among them. It then uses this information to construct a Boolean combination of key phrases for use as a search strategy by the DDC retrieval system. The problem of syntactic ambiguity is dealt with by a grammar which attempts to find the "most likely" parsing first and which enumerates additional parsings only if the first one proves to be uninterpretable. The syntactic representation produced by the parser is used by the semantic interpreter to construct a Boolean combination of keyphrases which is then simplified and converted to conjunctive normal form to approximate the input format of the DDC on-line retrieval system. For the purpose of the demonstration, this form is printed on the teletype instead of being sent directly to the retrieval component for execution.

In the complete on-line environment for which the pre-processor is envisaged, the DDC system would then perform thesaurus operations on these keyphrases, give the user a printout of the number of hits under each of his keyphrases, and perform the search. The user would thus have extensive feedback information for use in rephrasing his request for improved results. Although, the current system is oriented toward converting the Boolean expressions generated into conjunctive normal form, to agree with the formats which are accepted by the DDC system, this seems to introduce inefficiencies into the representation. It would be better to consider a retrieval system which could take arbitrary Boolean combinations and perform a retrieval based on the combination given, without restricting the format of the boolean combinations permitted.

6.2 Where We Stand

The target goals for this contract were the parsing and interpretation of 80% of an initial sample of 200 sentences and 75% of an additional trial sample during the final month of the contract. In terms of raw numbers of sentences parsed and interpreted, both of these levels have been met. However, the quality of the Boolean search strategies produced is still far below that which could be generated by a human analyst. Although the current contract has significantly advanced the state of the art in natural language processing, we do not currently have a capability to produce Boolean search strategies that would be adequate for a working system. We have however, developed the system to the point where parsing and interpretation of large numbers of requests is technically feasible, we have isolated the major problems which stand in the way of a practical implementation, and we have made contributions to the state of the art in natural language processing which will have wide applicability not only in the DDC environment, but in the wider context of the DOD in general and in the civilian

sector as well. One of these contributions has been the development of a grammar for a large subset of English within a framework which allows continual grammar development and the potential of approaching virtually unrestricted natural language input. This capability opens the way for applications well beyond the range of operations currently provided by the DDC on-line facility. (Examples would include the ability for a user to ask questions about the collection as a whole, about the system's capabilities, etc.) Also, the parsing of English requests opens the way for requests which include specific non-content restrictions on documents, such as contract number, supporting agency, etc. without the need for the user to learn elaborate and cumbersome formats. In addition, the parsing techniques that have been developed may have applications in the automatic standardization of phrasings in the mechanically generated indices produced by the DDC Machine Aided Indexing project, and in the machine aided construction of thesauruses.

6.3 Projections

A major objective of the current project was to obtain some estimate of the performance of a natural English querying facility in the DDC environment. Since there are a number of features of the testing environment of the current prototype which are significantly different from the intended application, some care is required in projecting the performance of the current prototype into an estimate of the performance of an eventual production system at DDC. In this section we will attempt to make such a projection on the basis of evidence gained from the current project.

6.3.1 Projected Performance

In discussing the performance of the prototype in Chapter 5, we have already discussed a number of sources of error or inadequacy in the current system's performance. Some of these are fundamental problems which will require further research. However,

there are others which are primarily functions of the short time period of the current project, and there are some that can be expected to change in an operational environment because of differences between that environment and the testing environment. One of the characteristics of the testing environment which can be expected to change is the nature of the requests themselves.

While the sample requests used for the testing of the prototype were drawn from the written requests which DDC receives by mail, the user of an on-line system will be typing the request directly into the machine on a keyboard and not writing it on paper or dictating it for his secretary. Thus, we should expect the requests to be shorter and more concise, and lacking some of the polite syntactic "sugar" such as "Please give me a list of references on the following topic: ...". The awareness that he is addressing a machine and not another human being will also tend to reinforce this behavior. Moreover, the samples received from DDC indicate already a strong tendency to simplify the requests to a single noun phrase describing the topic (although there is still a considerable volume of requests which are expressed as complete sentences). Of the sample of 200 requests which we received from DDC, 153 were noun phrases.

A second factor that will be operating, and one which cannot be underestimated in importance, is the reinforcement which the user will receive from the system and which will guide him into ways of negotiating with the system that are effective. Thus, we may expect the user's behavior to depend rather crucially on the responses which he gets from the system. The user will tend to use syntactic constructions which he has used before and which have been effective rather than new and strange constructions. This will be especially true after he has tried a few "strange" constructions and the system has failed to understand them. The task of the grammar designer for the system is to make the distinction between constructions which the system can handle and those which it will not as clear and intuitive to a naive user as possible. A grammar

which exhibits this characteristic is termbed "habitable"*.

With the awareness that he is talking to a machine and with the understanding that not all of English is acceptable, but only a habitable subset, we can expect the user of an on-line, natural language retrieval system to be considerably more cautious in the framing of his requests than he would be in writing a written request to DDC. In particular, we would expect a goodly number of sentences which we deemed ungrammatical in the initial sample and trial sample not to be uttered in an on-line environment. If we drop such sentences from consideration in our current samples, the performance rate on the initial sample rises to 162 out of 189 (or 86%).

An additional factor to be taken into account in projecting the performance onto a production system is the incomplete state of the present dictionary. We have seen in the post-trial processing of the trial sample that the correction of bugs and coding errors in the dictionary can appreciably improve the performance. Also we estimated that in the initial sample there were 10 sentences whose failure to parse appeared due to such dictionary errors. In a production system, we would expect such bugs to have been worked out of the dictionary so that these problems would not occur. Therefore taking this factor into account in addition to

---

*W. Watt (1968) discusses the problems associated with design of "habitable" languages:

"A "habitable language is one in which its users can express themselves without straying over the language's boundaries into unallowed sentences. If meant only to permit expressions addressed to a restricted subject matter, such a language can be a proper subset of, for example, the set of English sentences. Such a habitable subset is what any practical English man-computer interface must be. Yet, for any subject matter of interest, a subset that contains all English sentences "appropriate" to that subject will be impossibly large. Moreover, the subset must consist of the utterances of performance, not the sentences of competence; and besides having differences of form, utterances are partly interpreted and disambiguated on the basis of the situation usage. Thus the "English subset" must be redefined to bear the less direct relationship to the grammar that its performative nature indicates, and then, because of the large size of this subset, it must be greatly reduced, though without losing habitability."

the previous factor (i.e., with the addition of these 10 sentences and the elimination of the 11 bad ones), we would get a performance rate of 172 out of 189 (or 91%) on the initial sample. On the trial sample, including the post-trail processing and subtracting off ungrammatical sentences would give a performance rate of 81 out of 93 (or 87%). Thus, we could expect a projected performance rate slightly less than 90% from a production system based directly on the current grammar and semantic interpretation rules. Recall, however, that some of the requests which are included in this 90% figure are only marginally acceptable using the current grammar and rules.

Many other factors in projecting the performance of the current prototype onto that of a production on-line system are too nebulous to quantify. One of these is the fact that the adequacy of the first translated request is not the most important measure of performance in the production system. What is critical in this case is the adequacy of the request which results after several cycles of iteration during which the user revises and refines his request on the basis of the feedback from earlier requests. We have not had the opportunity in the current contract to investigate this sort of feedback in any detail, and can therefore say little about the effect of this factor other than to point out its existence. Additional factors which are obviously not quantifiable at the present are the degrees to which continued grammar development and refinement of the semantic interpretation rules will contribute to the improved quality of the Boolean requests generated.

6.3.2 Projected Execution Time

In projecting the timing figures quoted in section 5.2 onto an eventual production system on the Univac 1108 computer at DDC, there are a number of factors to be taken into account. The first of these is that the 1108 is approximately 2-2 1/2 times

faster than the PDP-10 on which these measurements were made . Another is that the LISP programming language which enabled this system to be developed in the short time frame of this contract pays for its flexibility with a significant overhead factor. Although this factor varies considerably depending on the situation, improvements by factors of 10-15 in operating speed are common in situations in which programs are converted from LISP to FORTRAN. Thus one should be able to expect something like a factor of 30 increase in speed in converting the current system to FORTRAN on the 1108, bringing the average processing time per sentence to the order of one second. Moreover, since the current system is designed mainly for developing the techniques for making natural English querying possible, and not for achieving the maximum possible speed, the speed improvement in a careful implementation could be considerably greater. Indeed, there is considerable room in the current LISP programs for speed improvements that could probably reduce the time required by another 10-20%.

6.3.3 Projected Cost-Effectiveness

In the current experimental prototype, the cost of processing an average sentence (30 seconds of computer time) costs approximately $3.00. This is the same order of magnitude as one might expect for human performance. (Assuming $12,000/year or $6.00/hour for the human analyst and approximately 20 minutes for an average request, the human indexing would cost $2.00 per request.) These figures are of course very crude estimates, but should be in the right ballpark (i.e. within a factor of 2). Thus, even the current system, which is designed as an experimental tool and not as an efficient production system, is roughly comparable in cost with a human indexer, although somewhat more expensive.

To extimate the cost for a production system on the Univac 1108, we can take our previous figure for the expected speed improvement from such an implementation (a factor of 30), and the fact that the gross cost of an 1108 is approximately 4 times that of the PDP-10 (and thus, the machine time should be approximately 4 times as expensive) to estimate a factor of something less than 8 reduction in cost per request (or a processing cost of something like 40¢ per request). This is sufficient to allow the user to make three or four requests to achieve a satisfactory result and still not exceed the cost of human processing.

Given that the cost of the machine system should not exceed the cost of human processing, what improvements/degradations in effectiveness might we expect? The current system is clearly not as sophisticated in the use of natural language as are the human analysts at DDC, but it has the advantage of being on-line with direct feedback to the user. Thus, the user has a chance through immediate feedback to become proficient with the on-line system in a way which is not open to him in the current manual operation (except in those cases where the analyst is forced to call him back for clarification, in which case, the cost of human processing goes well above the average figure which we assumed above). Also, the user gets his answers promptly, and since he has the possibility of asking several requests for the same cost as one in the manual operation, he can generally achieve a more satisfactory result by iteration than he could have achieved in the manual operation by mail.

The above projections, of course, depend on the ability of the system to adequately deal with the user's natural language input, so that the advantages of speed will not be offset by the necessity of making repeated requests before finding one that words. This will require considerable extension of the syntactic and semantic techniques used in the current research prototype.

Many of these extensions are straightforward operations of expanding the grammar and dictionary, correction of program bugs, etc., but others require some more research before a satisfying solution can be obtained. Nevertheless, we feel that an adequately useful, habitable subset of natural language could be devised which would make the on-line use of natural language querying a practical reality.

6.4 Conclusions

We feel that the performance levels attained by the current system after a development project of one year clearly demonstrate the feasibility of using natural English for querying an on-line retrieval system. Many of the limitations at the present time are largely due to the short time scale of the project; there are many changes which we could make which will greatly improve the performance statistics which we have simply not had sufficient time and resources within the current contract to make. With additional time for extending the grammar and the freedom to rule out difficult or confusing constructions to obtain a habitable subset, we should be able to develop a system with virtually 90% performance on the habitable subset (which would be very extensive). The mere caution that the user must use simple and clear grammatical English in the phrasing of his requests should go a long way toward improving the success ratio even for the system as it stands now.

Specific limitations of the current system and areas which require further work are as follows:

1. The diagnostic comments available to the user to indicate reasons why his sentence failed to parse are still much too limited.

2. The strategies for selecting the "most likely" parsing require additional work--especially for determining the scopes of reduced conjunctions.

3. The grammar requires about another year's development for including additional constructions, compacting the existing grammar, etc.

4. The system is currently implemented in the programming language LISP, which has facilitated the development of the system, but which introduces overhead that is undesirable in a production system. A production system should be implemented in a more efficient language. (e.g. FORTRAN on the 1108)

5. The system does not currently have any direct connection with the data base or thesaurus of the DDC system and is therefore hampered in its ability to generate keyphrases by the lack of knowledge of keyphrases which exist in the data base. The generation of keyphrases could be improved if this information were available. Also, this information could be useful in resolving syntactic ambiguities and determining the scopes of reduced conjunctions.

6. In the current system, the 80% performance specifications of the contract left us little room for experimenting with various habitible subsets of the language. It was necessary to write the grammar to deal with all of the constructions which occurred in the sample with any regularity (and some that occurred only once) without consideration of their impact on the rest of the grammar or the efficiency of the overall system. Before proceding to an operational implementation of such a system, it would be wise to obtain some evidence on the habitability of various subsets of the grammar.

6.5 Toward a Complete Natural English Preprocessor

As discussed above, the task of programming of a complete natural English preprocessor requires a number of prerequisites which have yet to be developed. There are probably several years' worth of research and development required at the current level of effort to obtain adequate solutions to the problems brought to light in the current study--determining the scopes of reduced conjunctions, providing adequate feedback to the user, using semantic information to select most likely parsings, etc. In addition, the grammar needs further development, and the semantic rules for computing the Boolean requests in the current system are only a crude approximation of what they should be to approach the skill of a human indexer. There is at least a man year's effort required in each of these areas. Also, for a complete system, a large dictionary would need to be constructed--on the order of 50-100,000 words. This is a clerical task of massive size, but relatively straightforward once the specifications for the dictionary entries are completely determined (this could be done by DDC staff). Complete determination of the dictionary

formats, however, would not be available until after the initial development period was complete. Also, after the initial period, a careful implementation of the system on the machine of ultimate use, with due attention to operating system environment and user characteristics, would be required. We believe this to be on the order of a 1-1 1/2 man-year effort. The necessary statistics on user performance and characteristics should be gathered during the research phase in order to make this implementation effective.

Thus, for the complete implementation of a system to translate natural English queries into Boolean requests suitable for input to the DDC retrieval system, at least two years are required at something like a 3-4 man level of effort each year, plus a large dictionary building effort during the second year. However, we feel that the full potential for natural language querying will not be gained from a system whose target language is limited to Boolean combinations of terms. The future of natural language querying, we feel, lies in the ability for the machine to understand the language of the document abstracts as well as that of the user's request to recognize distinctions for example between subject indicative information, and other types of information about a document, and to deal with each accordingly. The attainment of such an objective would require several years of additional research before systems with practical capabilities in these directions are feasible, but we believe that such systems can be realized and that they would provide considerably improved performance over existing capabilities--human or machine.

12

6.6 Future Possibilities

The real payoff of the use of natural language querying, we feel, lies in the extension of the retrieval system itself to permit additional possibilities for precision in the requests of the user. Currently the system is limited to providing only Boolean requests of subject indicative key phrases, with only a few suggestive uses of the equivalent of "role codes" for things such as author specification. Terms such as "technical report", "article", and "document" are all used as synonyms, although they could be intended by the user to have specific meanings. The use of natural language queries in conjunction with an expanded retrieval system could allow for the handling of requests with restrictions as to date, author, contracting agency, citations in other documents, level indicators (elementary, advanced, survey, etc.), journal, etc. without the need for elaborate coding conventions and formats--the normal linguistic constructions of the user's own natural language would suffice. In addition, natural English could be used to ask questions about the structure of the indexing system and the capabilities of the retrieval system as well as the specification of documents, thus permitting the on-line user to improve his performance through improving his proficiency with the system by means of questions which clarify its capabilities. The system would then be capable of educating its users for improved performance. This latter possibility, however, requires considerably more research.

6.7 A Recommendation for Future Contracts

We recommend that in future contracts of this type, the contractor (whoever he may be) should be given considerably more freedom to redirect his efforts (with consent of DDC) when it becomes apparent from the initial results that the original specifications of the project are inappropriate. The mechanical

understanding of natural language by computer is not an off-the-shelf capability which can be ordered with ironclad performance specifications, but is instead a field where performance of the type desired is slightly beyond the fringes of available technology and requires considerable research in the course of its attainment. In this particular instance, it became apparent in the course of the project that many of the provisions made in the original contract specifications were not appropriate and that the best interests of DDC and the government would be served if the effort was redirected. Specifically, we feel that the drive to obtain 80% performance on the sample of 200 sentences diverted effort from other more important aspects of the problem, and that there was good reason to believe that the sample on which we were working would not be representative of the requests of an on-line user. Likewise, it became apparent in the course of the contract that the system would be too large to operate in its entirety in the LISP system for the 1108 at DDC and that it benefited from the use of facilities which were not available in the 1108 LISP. Consequently, the investment of effort specifically to convert the programs from BBN-LISP to 1108 LISP would result in no real benefit and would be a waste of the government's money. However, our request to change this specification of the contract was refused and we were forced to divert effort from what we feel were important parts of the project to perform this conversion.

References

Bobrow, D.G., Murphy,D.P., and Teitelman, W.,"The BBN-LISP System", BBN Report 1677, Bolt Beranek and Newman Inc., Cambridge, Mass., April, 1968.

Chomsky, N., Aspects of the Theory of Syntax. Cambridge: M.I.T. Press, 1965.

Chomsky, N., Syntactic Structures. The Hauge: Mouton and Co., 1957.

Defense Documentation Center, DDC Remote On-Line Retrieval System Operator's Manual, DSA DDCM 4185.3 (Provisional), Defense Documentation Center, Alexandria, Virginia, May, 1970.

Myer, T.R. and Barnaby, J.R., TENEX Executive Language, Bolt, Beranek and Newman Inc., Cambridge, Mass., Jan., 1971.

Norman, E., "LISP Reference Manual for the UNIVAC 1108", University of Wisconsin Computing Center, Madison, Wisconsin, 1969.

Petrick, S., A Recognition Procedure for Transformational Grammars. Unpublished doctoral dissertation, M.I.T., 1965.

Stockwell, R., Schachter, and Partee, B., Integration of Transformational Theories on English Syntax. UCLA, 1968.

Watt, W.C., "Habitability", American Documentation, Vol. 19, No. 3, July 1968.

Woods, W.A., "Augmented Transition Networks for Natural Language Analysis", Harvard Computation Laboratory Report No. CS-1, Harvard University, Cambridge, Mass., Dec., 1969.

Woods, W.A., "Semantics for a Question-Answering System". Ph.D. thesis, Harvard University, Cambridge, Mass., Aug., 1967.

Woods, W.A., Transition Network Grammars for Natural Language Analysis. Communications of the ACM, 13, 591-602, Oct. 1970.

Woods, W.A., "Procedural Semantics for a Question-Answering Machine," AFIPS Conference Proceedings, Vol. 33 (1968, FJCC). (Enclosed as Appendix A).

Zwicky, A., Friedman, J., Hall, B., and Walker, D., "The MITRE Syntactic Analysis Procedure for Transformational Grammars", Proceedings of the Fall Joint Computer Conference, 1965, 317-326.

Appendix A.

THE DDC ENGLISH PREPROCESSOR USER'S GUIDE

April 1971

Negotiating with TENEX

The DDC English preprocessing system is implemented on the BBN-TENEX Time-Sharing System in Cambridge, Mass. In order to use the system, it is necessary to log into the TENEX system. This is done as follows: After establishing a telephone connection with the computer by dialing the computer's number from a data set or acoustically coupled teletype, the TENEX system will type something like:

```
BBN TENEX 1.21.00 5-APR-71 EXEC 1.28
@
```

The "@" sign is the TENEX executive's symbol which indicates that it is waiting for the user to type something. The user should now type:

```
LOGIN DDC
```

followed by a space, followed by a secret password (which will not print on the teletype), followed by another space, followed by an account number, followed by a carriage return. (The password and account number will be given to authorized users.) For a hypothetical account number 777777, the line on the teletype would look like:

```
@LOG DDC 777777
```

If you have logged in successfully, the system will type some information relating to your teletype and job number, and will type another "@" waiting for your input. If not, it will give an error comment and wait for you to try again. If you do not succeed in logging in within a reasonable period of time, the system will automatically log you out and break the telephone connection.

Once logged in, the user can call the execution of the DDC English preprocessor as described in the following section. He can return to the TENEX executive at any time by typing control C (i.e. by depressing the control key on the teletype and typing C). To log out of TENEX at the end of the session, return to the TENEX executive and type:

LOGOUT

followed by a carriage return. The system will type some accounting information and automatically break the telephone connection.

Note: If for some reason the telephone connection should be broken accidentally by some difficulty with the telephone line or for any reason other than a normal logout, the job will be held by the TENEX system in a "detached" status and can be resumed. This can be done by dialing up the machine again and instead of typing LOGIN, type ATTACH (space) (password) (space) (the job number that the system assigned you when you logged in)(carriage return). If you are successful, TENEX will type "@" with no further comment and you will be reattached to your old job. If you had lost the connection while in the English preprocessor subsystem, you can resume it by typing CONTINUE followed by a carriage return. If you have any trouble reattaching, call BBN by telephone. A detached job continues to be charged for computer hookup time until you reattach to it and log it out normally.

Using the English Preprocessor

After logging into the TENEX system, the user enters the English preprocessor by typing:

RUN DDCSYS

followed by a carriage return. The system will ther type a comment something like:

BBN LISP-10 11-31-70

and will then type a left arrow indicating that LISP is waiting for input. The user should then type:

TALKER()

to invoke the English preprocessor executive. Notice that LISP will echo a carriage return as soon as the parentheses in its input string ballance. TALKER will identify itself, and will then proceed to accept queries for processing or LISP commands for execution. The former consist of English sentences enclosed in parentheses, while the latter consist of LISP commands followed by arguments enclosed in parentheses.

TALKER indicates that it is waiting for input by typing its "system symbol", which consists of two asterisks.

To leave TALKER and return to the TENEX executive at the end of the session, one can either type control C as described before, or one can type the LISP command.

18

LOGOUT()

Control Characters in LISP

There are a number of special control characters which make life easy in the interactive LISP system in which the English processor runs. These characters are typed by depressing the control key on the teletype and typing the corresponding character. If printed on the teletype, a control character is preceeded by an upward arrow, however, most of the control characters do not print when they are typed, but cause a side effect. The following characters are useful.

Control A deletes the preceeding typed character. It indicates the deleted character by echoing a backwards slash followed by the deleted character.

Control Q deletes the current contents of the input buffer. (Generally the input buffer is the same as the current typed line--the exception being automatic carriage returns generated by the system when the user types beyond the end of the line. These exceptions are indicated by two asterisks beginning the new "line".)

Control R retypes the current contents of the input buffer (useful when echos from control A have made the current line unreadable).

Control D aborts whatever you are doing, and returns to the top level LISP executive. (useful whenever you get into trouble or want to discontinue a sentence and type another sentence. It throws you out of TALKER, however, and it is necessary to retype TALKER()).

Control E is less drastic than Control D and will usually abort a sentence and return to TALKER to await another sentence. It should usually be used before trying Control D. Control E will not break out of an embedded help loop, however.

Control C interrupts whatever you are doing and returns to the TENEX executive under which the LISP system runs. This is used in order to return to TENEX to logout, but can also be used to return to TENEX for any other reason. (The interrupted LISP system can be continued by typing

CONTINUE to the TENEX executive as long as it has not been supplanted by a call to some other subsystem.)

Control T can be used at any time to monitor the running status of your program and the time used. It is a useful control key to use to determine if the system is still alive (it doesn't respond to control T if it has stopped or crashed but it should respond promptly if the system is running.) and see how much time is being used (i.e. whether a sentence is taking a long time because the system is slow and you are not getting very much of the computer's time or because the parser is spending a lot of time trying to find a parsing). Control T causes immediate printing of the program's status (RUNNING, IO WAIT, etc.) followed by the number of hours, minutes, and seconds of cpu time and hook up time used since the user logged on.

Entering Queries for Processing

Normal English sentences or noun phrases can be entered by typing them to TALKER enclosed in parentheses. For example:

(GIVE ME INFORMATION ON THE HEATING OF ROCKETS BY PLUME IMPINGEMENT)

However, there are two special provisions to enable the user to give the equivalent of underlining or emphasizing key phrases. Since the teletype does not provide for upper/lower case distinctions or underlining, we have provided the following alternative: When a phrase of a sentence is to be indicated as a keyphrase it can be enclosed in parentheses with a preceeding up arrow or double up arrow (both have the same effect to the system). This tells the parser to consider the enclosed phrase as a proper noun phrase regardless of what the parentheses contain. This is a convenient way to include keywords or pharases which are unlikely to be in the dictionary of the system, such as names of authors or institutions, compound phrases that might be difficult for the system to parse, or esoteric jargon words.
For example:

(REFERENCES BY (↑ W.A. WOODS) ON (↑↑ NET2))

Depending on the setting of a number of mode variables, the system can display various intermediate results in the course of the processing. It can show the time spent in parsing and in interpreting, the parse tree that results, the intermediate semantic interpretation, and finally the Boolean request generated by the sentence. In the normal mode, only the final Boolean request will be displayed. However, any of the other displays can be obtained by setting the corresponding mode variable to "T" (the LISP system's symbol for "true") and they can be turned off again by setting the mode variable to "NIL" (the LISP system's equivalent

of "false"). For example, the commands:

```
SETQ(PPRINT T)
SETQ(ITIMEFLAG NIL)
```

will set the mode variable PPRINT to "T" indicating that parsings should be printed and will set ITIMEFLAG to "NIL" indicating that the interpretation time is not be be printed. The mode variables which govern these functions are PPRINT (print parsing), PRINT (print intermediate semantic interpretation), PTIMEFLAG (time the parsing), and ITIMEFLAG (time the semantic interpretation). There are other mode variables which govern other aspects of the system, but they are not necessary to the typical user.

Understanding the Output

The output Boolean request is an approximation of the input format required for the DDC WUIS system. It usually consists of an AND of either single phrases or OR's of single phrases, a form known to logicians as conjunctive normal form. The scopings of the AND's and OR's are indicated in the printout by parenthesis embeddings and by indentation. Since the natural interpretation of a user's query will rarely be in conjunctive normal form to begin with, the system automatically converts it. This relieves the user of the burden of making sure that his Boolean combination is in the correct form for the WUIS system, and in a system in which the query was passed automatically to the WUIS system without printout it would cause no confusion. However, when the user sees the conjunctive normal form expansion of his question, he will notice that many of his keyphrases appear several times in different components of the conjunction. This is the result of translating his request into a logically equivalent conjunctive normal form.

There are several other aspects of the printout that require explanation. The English processor never produces queries with NOT's in them unless the user explicitly uses a NOT in his query, and that event has been rare in the samples which we have seen. However, NOT's are possible in queries. The WUIS system places severe limitations on the use of NOT's in queries which it accepts, namely they can come only after all of the other components of the AND and they must be direct components of the AND. Wherever possible, the English processing system will convert its Boolean request to approximately this form. It indicates NOT's by embedding all of the positive components of the AND as the first component of a connective SDIFF (which stands for set difference) and all of the negative components in the second component of the SDIFF. The NOT's themselves do not appear. This form is more general than the WUIS system since

there are logical expressions that can be represented in it but not in the other. The request can be translated into the WUIS format only if the second component of the SDIFF is either a single phrase or an OR of single phrases. If it contains any AND's then the query is beyond the scope of the WUIS system.

The WUIS system also provides for special flags on keyphrases for emphasis, role codes, etc. One of these is appending an asterisk to a keyphrase to add emphasis. We have implemented a mechanism to add asterisks to keyphrases when the English sentence contains an indicator such as "especially" which suggests emphasis. We have also implemented a version of the author role code, which we have called "AUTHOR:" rather than the numeric field code used in the WUIS system. Finally, we have implemented a special indicator "VP:" to indicate keyphrases which are essentially verb phrases. These keyphrases are phrases which we felt might be useful content indicators but which would rarely exist in the DDC classification system at this time, but might exist in some future system. We therefore include it in the Boolean request, but leave it to the discression of the user whether to ignore it or to include some equivalent nominalization such as might be available from a synonym facility.

Encountering Unknown Words

The previous sections cover all of the basic information needed by a user as long as he uses only words that are in the vocabulary of the system, -- currently three or four thousand words. However, it is inevitable that a user, asking unconstrained questions about technical subjects will use words which the system has not previously encountered and are not in its vocabulary. We will therefore give here a brief description of the system's operation in that case.

When the system encounters a word which is not in its dictionary or is not an inflected form of a word in its dictionary (it performs inflectional morphology on the most common types of regularly inflected words), it announces this fact to the user with a comment:

```
I DON'T KNOW THE WORD XXXXXX
PLEASE TYPE ITS DICTIONARY ENTRY
D*
```

(where XXXXXX will be the word in question).

at this point, the system is in a special subsystem executive (which identifies itself with the system symbol "D*") waiting for the user to give it a dictionary entry for the indicated word. The user may at this point do any of several things. If he wants to give up on the sentence and try again from scratch, he can type QUIT followed by carriage return, and the system will return to ask for another sentence. If he knows the correct form for the needed dictionary entry, he can type DDEF followed by the proper dictionary entry to add the word to the dictionary, and then type OK followed by a carriage return to tell the system to continue its parsing. If the user does not know the proper format for

the dictionary entry, he can look at the dictionary entry for a similar word and copy it. The command DICT? followed by a word in parentheses will type the dictionary entry for a word. For example:

```
DICT?(REPORT)
```

would result in a typeout of the dictionary entry for the word "report", which would look like:

```
(REPORT
     N -S)
```

The dictionary entry will print out with indenting for easier reading, but the indenting is not necessary for a dictionary entry which the user types in. To type in this same dictionary entry, the user would type:

```
DDEF(REPORT N -S)
```

If the word which the system requests is the root form of the word or if it is an inflected form of a word which undergoes regular inflection, then the user need only give a dictionary entry for the root of the word. If, however, the word is an inflected form of a word which does not undergo regular inflection, then he should give entries for both the root word and the inflected form. The following interchange is an example:

```
I DON'T KNOW THE WORD MICE
PLEASE TYPE ITS DICTIONARY ENTRY
@DDEF(MOUSE N IRR)
MOUSE
@DDEF(MICE N (MOUSE (NUMBER PL)))
MICE
@OK
```

Here, the computer typed everything except the lines that begin with "@", and the computer typed the "@"'s which begin those lines. For more complete information on the format for dictionary entries, see the writeup on dictionary formats.

One frequent source of words which are not in the dictionary are misspelled words. If a misspelling is not caught by the user at the time he types it, the system will generate an "I DON'T KNOW THE WORD" comment and wait for a dictionary entry. If this is the case, the user can change the word to the correct word by typing CHANGEWORD followed by the new word or words enclosed in parentheses. For example:

```
I DON'T KNOW THE WORD MODOL
PLEASE TYPE ITS DICTIONARY ENTRY
D*CHANGEWORD(MODAL)
```

The system will respond by typing the remainder of the sentence to be parsed with the new substitution. Note that CHANGEWORD can be used to delete a word (by including no words in the parentheses) or to replace a single word by several (by including several words in the parentheses). Its use is in no way restricted to correcting spelling errors, however, and it can conveniently be used to substitute an equivalent word to see if the system knows it. (If not, the system will again respond with an "I DON'T KNOW THE WORD" message.) When the user is satisfied with the change, typing OK will cause the system to resume parsing on the changed string.

Logging Out

Although this information has already been covered in previous sections, I will cover it again here for easy reference. When the user has completed a session with the system and is ready to log out, he must first return to the TENEX executive. He can do this either by typing LOGOUT() or by typing control C. When the system responds with an "@", he need only type LOGOUT followed by a carriage return, and the system will automatically logout and break the telephone connection.

Appendix B.

The Transition Network Grammar

```
(PRINT (QUOTE "CREATED ") T)
(PRINT (QUOTE "13-JUN-71 13:54:58") T)
(TERPRI T)
  (PRINT (QUOTE DDCGRAMMARCOMS)
         T)
  (RPAQQ DDCGRAMMARCOMS ((P (COND
                              ((NEQ (CAR (QUOTE DDCGRAMMAR))
                                    (QUOTE NOBIND))
                                (MAPC DDCGRAMMAR
                                      (FUNCTION RPLACA)))))
           (G: DDCGRAMMAR)))
  (COND
    ((NEQ (CAR (QUOTE DDCGRAMMAR))
          (QUOTE NOBIND))
      (MAPC DDCGRAMMAR (FUNCTION RPLACA))))
(PRINT (QUOTE GRAMMAR:) T)
(PRINT (QUOTE DDCGRAMMAR) T)
(RPAQQ DDCGRAMMAR (COMPL/ COMPL/S COMPL/NTYPE FOR/FOR FOR/NP FOR/TO
ING/NP NP/ NP/, NP/,ESP NP/,NP NP// NP/ADV NP/ART NP/DET NP/HEAD
NP/HELPPRET NP/N NP/NP NP/NP: NP/ORD NP/QUANT NP/R NPR/ NPR/NPR
NPN/TITLE NPU/; NPU/;NP PAREN/ PAREN/PAREN PP/ PP/NP PP/PREP QUANT/
QUANT/QUANT QUANT/UNIT R/ R/NIL R/PREP R/WH S/ S/; S/;S S/AUX S/DCL
S/INF S/NP-SUBJ S/NP S/Q S/QDET S/SADV S/THERE S/VP VP/AGT VP/HEAD
VP/NP VP/V VP/VP))
(DEFINEQ


(COMPL/


          (* START OF COMPLEMENT NETWORK;
          TRANSFERS TO THE PROPER STATE FOR THE IDENTIFIED
          COMPLEMENTIZER.)


   (DO (COND
         ((WRD FOR)
           (TO FOR/FOR))
         ((WRD THAT)
           (SETRQ NTYPE THAT)
           (TO COMPL/NTYPE))
         (T (COND
              ((NULLR SUBJ)
                (SETR SUBJ (BUILDQ (NP (PRO SOMETHING))))))
            (JUMP FOR/NP))) ))


(COMPL/S

                                          (* LAST STATE OF
                                          COMPLEMENT NETWORK;
                                          POPS A COMPLEMENT NP
                                          STRUCTURE.)
   (POP (BUILDQ (NP + +)
                NTYPE S)
        T))
```

```
(COMPL/NTYPE

        (* LOOK FOR A COMPLETE S WHEN THE COMPLEMENTIZER
        (IN NTYPE) SO SPECIFIES)

   (PUSH S/ T
       (SETR S *)
       (TO COMPL/S)))

(FOR/FOR

        (* IF THE COMPLEMENTIZER IS 'FOR', LOOK FOR THE
        SUBJECT NP OF A FOR-TO COMPLEMENT)

   (PUSH NP/ T
       (SETR SUBJ *)
       (TO FOR/NP)))

(FOR/NP                                          (* LOOK FOR 'TO' OR 'NOT
                                                 TO'.)
   (WRD TO T
       (TO FOR/TO))
   (CAT NEG (NULLP NEG)
       (SETR NEG *)
       (TO FOR/NP)))

(FOR/TO

        (* IF 'TO', 'NOT TO', OR 'FOR' + NP WAS FOUND, LOOK
        FOR THE REMAINDER OF THE FOR-TO COMPLEMENT.)

   (PUSH VP/V (CHECKF V UNTENSED)
       (SENDR SUBJ (GETR SUBJ))
       (SENDR NEG (GETR NEG))
       (SENDR TNS (GETR TNS 1))
       (SENDRQ TYPE FOR-TO)
       (SETRQ NTYPE NOM)
       (SETR S *)
       (TO COMPL/S)))
```

```
(ING/BY

        (* IF THE SUBJECT WAS NOT PROPERLY DETERMINED IN A
        POSS-ING COMPLEMENT, LOOK FOR IT HERE.)

   (PUSH NP/ T
       (SETR SUBJ *)
       (TO VP/VP)))

(NP/
                                                (* START OF THE NP
                                                NETWORK.)
   (CAT DET T
       (COND

          (* IF THE DETERMINER IS A POSSESSIVE PRONOUN
          (MY, YOUR), CONSTRUCT THE POSSESIVE MODIFIER AND USE
          'THE' FOR THE DETERMINER)

         ((GETF POSSPRO)
           (ADDL ADJS (BUILDQ (POSS (NP (PRO *))))))
           (SETRQ DET THE))
         (T (SETR DET *)))
       (TO NP/ART))
   (CAT PRO T
       (SETR N (BUILDQ (PRO *)))
       (SETR NU (GETF NUMBER))
       (TO NP/NP))
   (NEW (WHETHER IF)
       T

          (* CONSTRUCT THE COMPLEMENT STRUCTURE FOR SENTENCES
          SUCH AS 'I DON'T KNOW WHETHER HE LEFT.')

       (SETR NTYPE *)
       (TO COMPL/NTYPE))
   (CAT NEG (NULL NEG)
       (SETR NEG *)
       (TO NP/))
   (JUMP NP/ART

          (* SINCE A PRONOUN OR DETERMINER IS NOT REQUIRED TO
          BEGIN AN NP, CONTINUE PROCESSING.)

       (NOT (CAT (DET PRO NEG))
            (WRD (WHOSE WHO WHAT WHETHER IF)))))
```

```
(NP/,                                          (* AFTER A COMMA AT THE
                                                END OF A NP.)

   (MEM (ETC. ETC)                             (* 'ETC' FILLS OUT A
                                                CONJOINED SERIES.)

       T
       (ADDR BODY *)
       (SETQ CONJ AND)
       (TO NP/,NP))
   (CAT ADV (RPEAT TRANSADV)                   (* A TRANSITIVE ADVERB
                                                ('PARTICULARLY',
                                                'ESPECIALLY') MAY
                                                INTRODUCE A POST-NOMINAL
                                                MODIFIER.)

       (SETR ADV *)
       (TO NP/,ESP))
   (CAT CONJ (NOR (WRD ,)                      (* A CONJUNCTION CAN
                  (GETR CONJ))                  LEAD INTO THE FINAL ITEM
                                                IN A SERIES.)

       (SETR CONJ *)
       (TO NP/,))
   (PUSH NP/ T                                 (* LOOK FOR THE NEXT
                                                ITEM IN THE SERIES.)

       (SENDR NPLIST T)
       (ADDR BODY *)
       (TO NP/,NP)))


(NP/,ESP                                       (* ANALYZE THE
                                                POST-NOMINAL MODIFIER
                                                INTRODUCED BY THE
                                                TRANSITIVE ADVERB)

   (PUSH NP/ T
       (ADDL NMODS (BUILDQ (ADVP (ADV +)
                                 *)
                           ADV))
       (COND
         ((NULLR PPFLAG)
           (SETQ PPFLAG T)))
       (TO NP/HEAD)))
```

```
(NP/,NP

        (* AN ITEM IN A COMMA-SPLICED NP SERIES HAS BEEN
        FOUND. IF THE NEXT WORD IS ',', LOOK FOR SUBSEQUENT
        ITEMS. OTHERWISE, POP THE CONJOINED STRUCTURE.)


    (WRD , T
        (T NP/,))
    (POP (COND
           ((AND (NULLP CONJ)
                 (WRD (AND OR)
                      (CADAR (LAST (GETR BODY)))))
            (PROG (BODY CONJ LAST TEMP)

           (* IF NO CONJ WAS FOUND AT THIS LEVEL BUT THE LAST
           NP IN THE SERIES WAS ITSELF A CONJOINED NP WITH A
           CONJ, PROMOTE THE CONJ UP TO THIS LEVEL:
           (NP1 NP2 (NP AND NP3 NP4)) -->
           (NP AND NP1 NP2 NP3 NP4))

                   (SETQ BODY (APPEND (GETR BODY)))
                   (SETQ LAST (LAST BODY))
                   (SETQ CONJ (CADR (SETQ TEMP (CAR LAST))))
                   (RPLACD LAST (CDDDR TEMP))
                   (RPLACA LAST (CADDR TEMP))
                   (RETURN (CONS (QUOTE NP)
                                 (CONS CONJ BODY)))))
           (T (BUILDQ (@ (NP #)
                         +)
                      (COND
                        ((GETR CONJ))
                        (T (QUOTE OR))

           (* IF THE LAST ITEM WAS NOT A CONJOINED NP AND THERE
           WAS NO CONJ AT THIS LEVEL, INSERT 'OR')


                        )
                      BODY)))
        (OP (GETR CONJ)
            (WRD (AND OR)
                 (CADAR (LAST (GETR BODY))))
            (NULL STRING))))
```

```
(NP//
                                                    (* FIND THE SECOND TERM
                                                    IN A RATIO)
   (CAT N T
       (SETR N (BUILDQ (N + / (N *))
                       N))
       (TO NP/DET))
   (PUSH NPR/ T
       (SETR N (BUILDQ (N + / *)
                       N))
       (TO NP/DET)))


(NP/ADV

          (* AN ADVERB HAS BEEN FOUND AFTER THE DETERMINER
          STRUCTURE HAS BEEN BUILT, OR AFTER 1 OR MORE
          PRENOMINAL MODIFIERS HAS BEEN PROCESSED.
          A SEQUENCE OF ADDITIONAL ADVERBS IS ALLOWED, UNTIL
          THE ADJECTIVE THEY MODIFY IS FOUND, COMPLETING THIS
          PARTICULAR PRENOMINAL MODIFIER.)


   (CAT ADJ T
       (ADDL ADJS (BUILDQ (@ (ADJP)
                             #
                             ((ADJ *)))
                          (REVERSE (GETR ADVS))))
       (TO NP/DET))
   (CAT ADV T
       (ADDL ADVS (BUILDQ (ADVS *)))
       (TO NP/ADV)))


(NP/ART

          (* AN ARTICLE (POSSIBLY NULL) HAS BEEN FOUND;
          LOOK FOR AN OPTIONAL ORDINAL MODIFIER
          (POSSIBLY A SUPERLATIVE ADJECTIVE))


   (CAT ORD T
       (SETR POSTART (BUILDQ ((ORD *))))
       (TO NP/ORD))
   (CAT ADJ (GETP SUPERLATIVE)
       (SETR POSTART (BUILDQ ((ORD SUPERLATIVE *))))
       (TO NP/ORD))
   (JUMP NP/ORD 2))
```

```
(NP/DET

        (* HERE AFTER THE COMPLETE DETERMINER STRUCTURE
        (INCLUDING ART, ORD, AND QUANT) HAS BEEN PROCESSED.
        LOOK FOR POSSIBLE PRENOMINAL MODIFIERS
        (ADJECTIVES OR PARTICIPLES
        (WITH ADVERBS)) AND THEN LOOK FOR A POTENTIAL
        HEAD--AN N, NPR, OR GERUND, OR EVEN A POSS-ING
        NOMINALIZATION.)


   (CAT ADJ T
        (ADDL ADJS (BUILDQ (@ (ADJ)
                              #                    (*)
                              )
                           FEATURES))
        (TO NP/DET))
   (CAT N T
        (SETR N (BUILDQ (N *)))
        (SETR NU (GETF NUMBER))
        (TO NP/N))
   (CAT ADV T
        (SETR ADVS (BUILDQ ((ADVS *))))
        (TO NP/ADV))
   (CAT V (OR (GETF PASTPART)
              (GETF PRESPART))
                                                  (* PRENOMINAL
                                                  PARTICIPLES)
        (ADDL ADJS (BUILDQ (ADJ (PARTICIPLE #))
                           LEX))
        (TO NP/DET))
   (CAT V (GETF PRESPART)
                                                  (* GERUND HEAD, AS IN
                                                  'FREEZE DRYING')
        (SETR N (BUILDQ (N #)
                        LEX))
        (SETQ NU SG)
        (TO NP/N))
   (PUSH S/AUX (OR (CAT (NEG ADV))
                   (CHECKF V PRESPART))


        (* PUSH FOR A POSS-ING NOMINALIZATION.
        ('JOHN'S FALLING ...') IF A POSSESSIVE MODIFIER HAS
        BEEN FOUND, IT BECOMES THE SUBJECT OF THE COMPLEMENT
        SENTENCE, OTHERWISE THE SUBJECT IS 'SOMETHING')
```

```
        (SENDR SUBJ (COND
                ((WRD POSS (CADR (GETR ADJS)))
                  (SETQ TEMP (CADAR (GETR ADJS)))
                  (SETR ADJS (CDR (GETR ADJS)))
                  TEMP)
                (T (SENDR SUBFLAG T)
                   (BUILDQ (NP (PRO SOMETHING))))))
        (SENDRQ TYPE POSS-ING)
        (SETRQ NTYPE NOM)
        (SETR S *)
        (TO COMPL/S))
    (PUSH NPR/ T
        (SETR N *)
        (SETRQ NU SG)
        (TO NP/N)))

(NP/HEAD

        (* HERE WHEN THE HEAD OF THE NP HAS BEEN POSITIVELY
        DETERMINED. LOOK FOR POST-NOMINAL MODIFIERS:
        PREPOSITIONAL PHRASES, RELATIVE CLAUSES, TO- AND
        THAT- COMPLEMENTS. PPFLAG IS T AFTER A PP HAS BEEN
        FOUND; IT INSURES THAT SUBSEQUENT REDUCED RELATIVES
        WILL MODIFY THE NEAREST NP.)

    (PUSH R/ (OR (WRD (WHO WHOM WHOSE WHICH THAT))
                 (AND (WRD (WHICH WHOM WHOSE)
                           (NEXTWRD))
                      (CAT PREP)))
        (SENDRQ TYPE REL)
        (SENDR WH (BUILDQ (NP (DET WHR)
                              +
                              (NU +))
                          N NU))
        (ADDL NMODS *)
        (TO NP/R))
    (PUSH PP/ (CAT PREP)
        (ADDL NMODS *)
        (COND
          ((NULLR PPFLAG)
            (SETRQ PPFLAG T)))
        (TO NP/HEAD))
    (PUSH FOR/NP (WRD TO)
                                        (* LOOK FOR
                                        TO-COMPLEMENTS: 'THE WAY
                                        TO DO IT...')
        (ADDL NMODS (BUILDQ (COMPL *)))
        (COND
          ((NULLR PPFLAG)
            (SETRQ PPFLAG T)))
        (TO NP/HEAD))
```

```
    (TST R/NIL (AND (GETR PPFLAG)
                    (NOR (WRD (WHAT WHO WHOM WHICH THAT WHOSE))
                         (GETR QDET)
                         (WRD BE (GETROOT * V))))
        (SUSPEND 1)

           (* LOOK FOR REDUCED RELATIVES AFTER SEEING A PP.
           THE SUSPEND MEANS THAT THE ACTIONS
           (INCLUDING THE PUSH) WILL GET DONE AFTER THE
           FOLLOWING JUMP ARC HAS BEEN TAKEN AND LEADS TO A
           BLOCK; THE SUBSEQUENT BACKUP WILL CAUSE THE RELATIVE
           IN 'THE MAN NEAR THE GIRL I SEE' TO MODIFY 'GIRL'
           INSTEAD OF 'MAN' IN THE FIRST PARSE.)

        (SENDRQ TYPE REL)
        (SENDR WH (BUILDQ (NP (DET WHR)
                              +
                              (NU +))
                          N NU))
        (PUSH R/NIL)
        (ADDL NMODS *)
        (T NP/R))
    (JUMP NP/NP T

           (* THIS SETS UP THE REGISTER NPFEATURES AT THE LEVEL
           ABOVE THIS SO THAT THE RESUME MACHINERY WILL RETURN
           AN EXTRAPOSED RELATIVE CLAUSE TO THIS POSITION)

        (LIFTR NPFEATURES (RESUMETAG NP/HEAD)))
    (PUSH R/NIL (NOR (GETR PPFLAG)
                     (WRD (WHAT WHO WHOM WHICH THAT WHOSE))
                     (GETR QDET)
                     (WRD BE (GETROOT * V)))
        (SENDRQ TYPE REL)
        (SENDR WH (BUILDQ (NP (DET WHR)
                              +
                              (NU +))
                          N NU))
        (ADDL NMODS *)
        (T NP/R))
    (PUSH COMPL/

           (* FOR NOUNS MARKED FACTN (E.G. 'STATEMENT', 'FACT'),
           THIS ARC LOOKS FOR A THAT-COMPLEMENT
           ('THE FACT THAT I ARRIVED...'))
```

```
        (AND (WRD THAT)
             (WRD (A THE)
                  (CADR (GETR DET)))
             (RFEAT FACTN HEAD))
        (ADDL NMODS (BUILDQ (COMPL *)))
        (T  NP/NP))
    (PUSH COMPL/NTYPE                          (* LOOK FOR A
                                               THAT-COMPLEMENT WITH
                                               DELETED 'THAT')

        (RFEAT FACTN HEAD)
        (SENDRQ NTYPE THAT)
        (ADDL NMODS (BUILDQ (COMPL *)))
        (TO NP/NP)))


(NP/HELDPART                                   (* A PARTITIVE HAS BEEN
                                               TAKEN OFF THE HOLD LIST
                                               AT STATE NP/QUANT.)
    (JUMP NP/DET T                             (* LOOK FOR A REGULAR
                                               HEAD: 'OF THESE HOW MANY
                                               MEN WENT ...'))
    (JUMP NP/HEAD T

        (* THE HEAD WAS DELETED; INSERT AN APPROPRIATE
        DUMMY: 'OF THESE HOW MANY WENT ...')

        (SETRQ N (PRO ONES))
        (SETRQ NU SG/PL)))


(NP/N

        (* A TENTATIVE HEAD HAS BEEN FOUND, BUT IT MAY BE
        ONLY THE FIRST PART OF A NOUN-NOUN OR
        NOUN-ADJECTIVE-NOUN SEQUENCE.)

    (CAT LIST (AND (WRD SG NU)                 (* ADJUST NU FOR AN
                                               ALTERNATIVE PLURAL
                                               SPECIFICATION 'BOY
                                               (S) ')
                   (WRD (S ES)
                        (CAR *)))
        (SETRQ NU SG/PL)
        (TO NP/N))
    (WRD / T

        (* '/' FOLLOWS THE TENTATIVE HEAD, INDICATING THAT
        IT WAS THE FIRST TERM OF A RATIO)
```

```
    (TO NP//))
(CAT POSS T

        (* POSS ('S) MARKS THE PRECEDING HEAD AS A GENITIVE
        MODIFIER ON A HEAD WHICH IS TO FOLLOW.
        SET UP THE PROPER STRUCTURE AND LOOP TO NP/DET.)

    (SETR ADJS (BUILDQ ((POSS #))
                       (NPBUILD)))
    (SETRQ DET THE)
    (TO NP/DET))
(CAT N (AND (NOT (WRD PL NU))
            (SEMNET (CADR (GETR N))
                    *))

        (* A NEW HEAD IS FOUND, IMPLYING THAT THE PRECEDING
        HEAD IS A NOUN-MODIFIER.)

    (ADDL ADJS (BUILDQ (ADJ +)
                       N))
    (SETR N (BUILDQ (N *)))
    (SETR NU (GETF NUMBER))
    (TO NP/N))
(PUSH NPR/ T                                    (* THE NEW HEAD IS A
                                                PROPER NOUN MODIFIED BY
                                                THE OLD HEAD.)

    (ADDL ADJS (BUILDQ (ADJ +)
                       N))
    (SETR N *)
    (SETRQ NU SG)
    (TO NP/N))
(CAT ADJ (AND (NOT (WRD PL NU))
              (SEMNET (CADR (GETR N))
                      *))
                                                (* AN ADJECTIVE AFTER A
                                                TENTATIVE HEAD IMPLIES
                                                AN N-ADJ-N STRUCTURE.)

    (ADDL ADJS (BUILDQ (ADJ +)
                       N))
    (ADDL ADJS (BUILDQ (@ (ADJ)
                          #                     (*)
                          )
                       FEATURES))
    (TO NP/DET))
(CAT V (AND (GETF PRESPART)
            (NOT (WRD PL NU))
            (NOT (VPARTICLE * (NEXTWRD))))

        (* A GERUND HERE IS TAKEN AS THE HEAD--A SUBSEQUENT
        N WILL MOVE IT TO A PARTICIPIAL MODIFIER POSITION.)
```

```
        (ADDL ADJS (BUILDQ (ADJ +)
                           N))
        (SETR N (BUILDQ (N #)
                        LEX))
        (SETRQ NU SG)
        (TO NP/N))
    (JUMP NP/HEAD T
        (SETR HEAD (CADR (GETR N))))
    (CAT N (WRD PL NU)
```

(* A SPECIAL ARC TO HANDLE N-N MODIFIERS WHERE THE FIRST NOUN IS PLURAL: 'OPERATIONS RESEARCH', 'SYSTEMS ANALYSIS'. THIS MIGHT NOT BE A PRODUCTIVE PROCESS, IN WHICH CASE THIS ARC IS UNNECESSARY AND SHOULD BE REPLACED BY APPROPRIATE COMPOUND DICTIONARY ENTRIES.)

```
        (ADDL ADJS (BUILDQ (ADJ +)
                           N))
        (SETR N (BUILDQ (N *)))
        (SETR NU (GETF NUMBER))
        (TO NP/N)))

(NP/NP
```

(* THE ANALYSIS OF THE CURRENT NP HAS BEEN ESSENTIALLY COMPLETED. A COMMA AT THIS POINT CAN SIGNIFY THAT THIS IS THE BEGINNING OF A SERIES OF CONJOINED NP'S (ARC 1), WHILE A PARENTHETIC EXPRESSION (A LIST) IS INTERPRETED AS A NON-RESTRICTIVE MODIFIER ON THIS NP (E.G. 'FIBROUS MATERIALS (ASBESTOS, FIBERGLASS) '). A COLON CAN ALSO INDICATE THE BEGINNING OF A SERIES OF NON-RESTRICTIVE ITEMS (ARC 3). THE NORMAL CASE, HOWEVER, IS TO POP THE NP SO FAR ANALYZED.)

```
    (WRD , (NULLR NPLIST)
```

(* THIS ARC STARTS OFF A SERIES OF COMMA-CONJOINED NP'S, WHICH ARE ANALYZED BY PUSHING FOR NP'S FROM WITHIN THE FIRST NP OF THE SERIES. NPLIST IS ONLY EMPTY FOR THE TOP-LEVEL NP (THE FIRST ONE) OF THE SERIES, SO THAT ALL NP PUSHES ARE DONE FROM THE TOP LEVEL, THAT IS, THE SECOND NP CAN'T PUSH FOR THE THIRD, THE THIRD FOR THE FOURTH, ETC. AT THE TOP-LEVEL, THE SUBSEQUENT ITEMS IN THE SERIES ARE COLLECTED IN THE REGISTER BODY.)

```
        (SETR BODY (LIST (NPBUILD)))
        (TO NP/,))
    (CAT LIST T

          (* THIS IS A TRICKY ARC--IT RECURSIVELY CALLS THE
          PARSER TO ANALYZE THE LIST OF NON-RESTRICTIVE
          MODIFIERS, BEGINNING AT STATE PAREN/ IN THE GRAMMAR.
          THE PARSE IS ADDED TO NR, AND IF IT IS NIL, WE
          ABORT.)

        (ADDL NR (PARSE?PARENS * PAREN/))
        (COND
          ((NULL (CAR (GETR NR)))
            (ABORT)))
        (TO NP/HEAD))
    (WRD : T
                                                (* THIS ARC HANDLES
                                                'INFORMATION ON THE
                                                FOLLOWING: RADAR,
                                                LASERS...')
        (TO NP/NP:))
    (POP (NPBUILD)
        T))

(NP/NP:

          (* CURRENTLY, THE ONLY POSSIBILITY AFTER A COLON AT
          THE END OF A NP IS ANOTHER NP
          (PERHAPS A CONJUNCTION OF NP'S))

    (PUSH NP/ T
        (ADDL NR *)
        (TO NP/NP)))

(NP/ORD

          (* AN ORDINAL INDICATOR, IF PRESENT, HAS BEEN
          ANALYZED. AN OPTIONAL QUANTIFIER CAN FOLLOW: 'FIVE
          MEN', 'MANY PLANES'. THE QUANTIFIER IS ADDED TO THE
          POSTARTICLE STRUCTURE.)

    (PUSH QUANT/ (CAT (QUANT INTEGER ADV COMP))
        (SETR POSTART (.WILDQ (* + *)
                              POSTART))
        (TO NP/QUANT))
    (JUMP NP/QUANT T))
```

```
(NP/QUANT

          (* THE QUANTIFIER OR ORDINAL
          (AND CERTAIN DETERMINERS) CAN BE FOLLOWED BY A
          PARTITIVE CONSTRUCTION, AS IN 'THE LAST OF THE
          MOHICANS', 'FIVE OF THE BOYS', OR 'HOW MANY OF THE
          DOCUMENTS...'. THE PARTITIVE IS USUALLY INTRODUCED
          BY THE PREPOSITION 'OF' (ARC 2), BUT FOR SOME
          DETERMINERS (E.G. 'ALL', 'BOTH') THE PREPOSITION MAY
          BE MISSING (ARC 2); ARC 3 RETRIEVES A PARTITIVE THAT
          WAS ANALYZED AND PUT ON THE HOLD LIST AT STATE
          S/--'OF THE BOYS HOW MANY ...'.
          WHEN THERE IS A PARTITIVE, IT IS ADDED TO THE LIST
          OF NOUN MODIFIERS, AND THE HEAD OF THE NP BECOMES
          THE DUMMY ELEMENT 'ONES'.)

   (PUSH PP/ (AND (WRD OF)
                  (OR (GETR POSTART)
                      (WRD (WHICHQ HOWMANY HOWMUCH ALL SEVERAL)
                           DET)))
       (ADDL NMODS *)
       (SETR DET (DETBUILD))
       (SETRQ N (PRO ONES))
       (SETRQ NU SG/PL)
       (TO NP/HEAD))
   (PUSH PP/PREP (WRD (ALL BOTH)
                      DET)
       (SENDRQ PREP OF)
       (ADDL NMODS *)
       (SETR DET (DETBUILD))
       (SETRQ N (PRO ONES))
       (SETRQ NU SG/PL)
       (TO NP/HEAD))
   (VIR PP (AND (GETF PARTITIVE)
                (OR (GETR POSTART)
                    (WRD (WHICHQ HOWMANY HOWMUCH ALL SEVERAL)
                         DET)))
       (SETR DET (DETBUILD))
       (ADDL NMODS *)
       (TO NP/HELDPART))
   (JUMP NP/DET T
       (SETR DET (DETBUILD))))

(NP/R

          (* A RELATIVE CLAUSE HAS BEEN ANALYZED.
          THIS MAY BE FOLLOWED (OPTIONALLY) BY ANOTHER FULL
          RELATIVE (NOT REDUCED).)
```

```
   (PUSH R/ (WRD (WHO WHOM WHICH THAT))
       (SENDRQ TYPE REL)
       (SENDR WH (BUILDQ (NP (DET WHR)
                             +
                             (NU +))
                         N NU))
       (ADDL NMODS *)
       (T NP/R))
   (JUMP NP/NP T))


(NPR/

          (* START OF THE PROPER NOUN NETWORK.
          EVENTUALLY, THIS WOULD INCLUDE A FULL GRAMMAR FOR
          THE SYNTAX OF PROPER NAMES--TITLES, ABBREVIATIONS,
          INITIALS, ETC. CURRENTLY, WE RECOGNIZE CERTAIN WORDS
          AS TITLES IF THEY ARE FOLLOWED BY A WORD IN THE NPR
          CATEGORY; OTHERWISE, THIS NETWORK WILL ONLY
          RECOGNIZE ISOLATED NPR WORDS AS PROPER NOUNS.)


   (MEM (SAMPLE ROCK LINE LINE# APOLLO)
       T
       (SETR TITLE *)
       (T NPR/TITLE))
   (CAT NPR T
       (SETR NPR (LIST *))
       (T NPR/NPR)))


(NPR/NPR

          (* END OF THE PROPER NOUN NETWORK.
          ARC 1 POPS THE APPROPRIATE STRUCTURE;
          ARC 2 INSURES THAT THE SYSCONJ FACILITY WILL NOT BE
          INVOKED AT THIS LEVEL, THAT IS, THAT A CONJUNCTION
          OF NPR'S WILL BE ANALYZED AS A CONJUNCTION OF NP'S
          WITH NPR HEADS, NOT AS A SINGLE NP WITH A CONJOINED
          NPR HEAD.)


   (POP (BUILDQ (& (NPR)
                   +)
                NPR)
       T)
   (CAT CONJ NIL))
```

```
(NPR/TITLE

          (* HERE IF A TITLE WORD WAS FOUND.
          PICK UP THE FOLLOWING NPR AND BUILD THE CORRECT
          STRUCTURE.)

   (CAT NPR T
       (SETR NPR (BUILDQ (+ *)
                         TITLE))
       (TO NPR/NPR)))


(NPU/;

          (* HERE IF A ';' WAS FOUND IMMEDIATELY AFTER THE
          SUBJECT NP. THIS MARKS THE SENTENCE AS A NOUN-PHRASE
          UTTERANCE CONSISTING OF A SEQUENCE OF CONJOINED
          NP'S. (SEMI-COLON CONJOINING IS NOT ALLOWED WITHIN
          THE NP'S OF A REGULAR SENTENCE.) THE STRATEGY HERE
          IS SIMILAR TO THAT USED FOR SEMI-COLON CONJUNCTION
          AT THE END OF FULL SENTENCES
          (STATES S/; AND S/;S) AND SOMEWHAT RESEMBLES THE
          OPERATION OF COMMA-CONJOINING WITHIN NP'S
          (STATES NP/, AND NP/,NP))

   (CAT CONJ (OR (NULLR CONJ)
                 (EQ * (GETR CONJ)))
       (COND
         ((NULLR CONJ)
          (SETR CONJ *)))
       (T  NPU/;))
   (PUSH NP/ T
       (ADDL NPU *)
       (TO NPU/;NP)))


(NPU/;NP

          (* A SEQUENCE OF SEMICOLON-CONJOINED NP'S IN AN NPU
          CAN BE FOLLOWED BY A ';', INDICATING THAT ANOTHER
          ITEM IS TO FOLLOW, OR ELSE THE END OF THE STRING
          MUST HAVE BEEN REACHED. IN WHICH CASE THE FINAL NPU
          STRUCTURE IS BUILT.)
```

```
   (WRD ; T
       (TO NPU/;))
   (POP (BUILDQ (S NPU (@ (NP #)
                          #))
                (COND
                  ((GETR CONJ))
                  (T (QUOTE OR)))
                (REVERSE (GETR NPU)))
       (NULL STRING)))


(PAREN/

          (* THIS IS THE INITIAL STATE FOR THE GRAMMAR WHICH
          ANALYZES POST NOMINAL PARENTHETIC EXPRESSIONS.
          CURRENTLY, ONLY A NOUN-PHRASE CAN OCCUR AS SUCH A
          NON-RESTRICTIVE MODIFIER, BUT THE GRAMMAR SHOULD BE
          EXPANDED HERE TO INCLUDE SEQUENCES OF ADJECTIVAL
          PHRASES.)


   (PUSH NP/ T
       (SETR PAREN *)
       (TO PAREN/PAREN)))


(PAREN/PAREN

          (* THE FINAL STATE OF THE PARENTHETIC-EXPRESSION
          GRAMMAR; JUST POP WHATEVER WAS IDENTIFIED.
          OF COURSE, WE MUST HAVE EXHAUSTED THE STRING WITHIN
          THE PARENTHESES.)


   (POP (GETR PAREN)
       T))


(PP/

          (* FIRST STATE OF THE PREPOSITIONAL PHRASE NETWORK.
          ALL PUSHES TO THIS STATE MAKE SURE THAT THE CURRENT
          WORD IS A PREPOSITION, SO WE CAN OMIT THE TEST
          HERE.)


   (TO (SETR PREP *)
       (TO PP/PREP)))
```

```
(PP/NP

        (* HERE AFTER THE PREP AND NP HAVE BEEN FOUND,
        THE PP STRUCTURE IS BUILT AND SPOPPED, THAT IS,
        POPPED TO THE LEVEL DETERMINED BY THE SELECTIVE
        MODIFIER PLACEMENT FACILITY.
        SINCE THE DICTIONARY DOES NOT YET CONTAIN SMP
        FEATURES, THE DEFAULT PLACES THE PP IN THE LOWEST
        CONSTITUENT IT CAN BELONG TO.)

   (SPOP (BUILDQ (PP (PREP +)
                     +)
                 PREP NP)
         T))

(PP/PREP
                                        (* AFTER PICKING UP THE
                                        PREP, FIND THE NP
                                        PREPOSITIONAL OBJECT.)
   (WRD : T

        (* IF THE PREP IS FOLLOWED BY ':', SKIP PAST IT AND
        LOOK FOR THE OBJECT IN THE REGULAR WAY.
        E.G. 'INFORMATION ON: RADAR, LASERS...' IS PROPERLY
        ANALYZED IF THE COLON IS IGNORED.)

        (TO PP/PREP))
   (PUSH NP/ T

        (* NORMALLY, THE OBJECT OF THE PREP WILL BE FOUND AS
        AN ORDINARY NP STARTING AT THIS STRING POSITION.)

        (SETR NP *)
        (TO PP/NP))
   (VIR NP T

        (* THE OBJECT MIGHT HAVE BEEN FRONTED, FOR EXAMPLE,
        BY RELATIVIZATION OR PASSIVIZATION, LEAVING A
        DANGLING PREPOSITION ('THE STORE I BOUGHT IT IN
        ...'); IF SO, THE OBJECT HAS BEEN PLACED ON THE HOLD
        LIST BY PREVIOUS STATES, AND THIS ARC RETRIEVES IT.
        THE RESUME ACTION IS NECESSARY TO DEAL WITH A
        RELATIVE CLAUSE EXTRAPOSED FROM THE FRONTED OBJECT
        AND LEFT IN THIS POSITION ('THE STORE I BOUGHT IT IN
        WHICH USUALLY HAS GOOD PRICES...'))
```

```
        (RESUME)
        (SETR NP *)
        (TO PP/NP))
    (VIR ADV (AND (WRD WHERE (CADR *))
                  (WRD (FROM TO AT)
                       PREP))

          (* IF THE DANGLING PREP IS A LOCATIVE ONE AND THE
          WORD 'WHERE' WAS FOUND AND HELD BY PREVIOUS STATES
          (E.G. S/) WE RETRIEVE IT HERE AND BUILD THE
          APPROPRIATE PP STRUCTURE.)

        (SETR NP (BUILDQ (NP (DET WHQ)
                             (N PLACE)
                             (NU SG))))
        (TO PP/NP))
    (VIR ADV (AND (WRD WHEN (CADR *))
                  (WRD AT PREP))

          (* IF WE HAVE A DANGLING TEMPORAL PREP AND WE
          PREVIOUSLY ENCOUNTERED AND HELD 'WHEN', WE BUILD A
          TIME PP.)

        (SETR NP (BUILDQ (NP (DET WHQ)
                             (N TIME)
                             (NU SG))))
        (TO PP/NP)))

(QUANT/

          (* START OF QUANTIFIER NETWORK FOR NP DETERMINER
          STRUCTURE. QUANTIFIER CAN INCLUDE A COMPARATIVE
          ('MORE THAN') OR ELSE JUST BEGIN WITH AN INTEGER OR
          A WORD IN CATEGORY QUANT. CURRENTLY, MOST WORDS IN
          THIS CATEGORY ARE ALSO IN CATEGORY DET, SO THEY
          APPEAR AS DETERMINERS IN THE FIRST PARSES.)

    (CAT COMP (NULL ADV)
        (SETR ADV *)
        (TO QUANT/))
    (CAT QUANT T
        (SETR NUMB *)
        (TO QUANT/QUANT))
    (CAT INTEGER T
        (SETR NUMB *)
        (TO QUANT/QUANT)))
```

```
(QUANT/QUANT

        (* AFTER THE QUANTIFIER HAS BEEN PICKED UP, A UNIT
        OF MEASURE CAN BE SPECIFIED
        ('FIVE GALLONS', 'MORE THAN 3 MM'). POTENTIAL UNITS
        HAVE A UNIT MARKER, WHICH IS TESTED ON THE ARCS FROM
        THIS STATE.)


   (TST UNIT-TST (MARKER UNIT *)
        (SETR UNIT *)
        (SETRQ FLAG MUCH)
        (TO QUANT/UNIT))
   (JUMP QUANT/UNIT (NOT (MARKER UNIT *))
        (SETRQ FLAG MANY)))

(QUANT/UNIT

        (* END OF QUANTIFIER NETWORK;
        BUILD THE CORRECT STRUCTURE.
        IF A UNIT IS PRESENT, THE STRUCTURE IS FLAGGED WITH
        THE WORD 'MUCH', OTHERWISE 'MANY'.
        ALSO, IF THERE WAS A COMPARATIVE, THE ROOT OF THE
        QUANTIFIER STRUCTURE IS THE NODE 'COMP'.)

   (POP (COND
        ((GETR ADV)
          (BUILDQ ((COMP (ADV +)
                         (@ (NP (INTEGER +))
                            #))
                   +)
                  ADV NUMB (COND
                    ((GETR UNIT)
                      (BUILDQ ((UNIT +))
                              UNIT))
                    (T NIL))
                  FLAG))
        (T (BUILDQ ((@ (NP (INTEGER +))
                       #)
                    +)
                   NUMB
                   (COND
                     ((GETR UNIT)
                       (BUILDQ ((UNIT +))
                               UNIT)))
                   FLAG)))
     T))
```

```
(R/

          (* START OF RELATIVE CLAUSE NETWORK, GIVEN THAT WE
          ARE LOOKING AT A RELATIVE PRONOUN
          ('WHO', 'WHAT', 'WHICH') OR A PREP FOLLOWED BY A
          RELATIVE PRONOUN.)

   (MEM (WHICH THAT WHO)
        T
        (TO R/WH))
   (WRD WHOM T

          (* FOR 'WHOM', WE KNOW THAT THE WH-NP IS NOT THE
          SUBJECT OF THE RELATIVE CLAUSE--WE HOLD IT TO BE
          PICKED UP LATER.)

        (HOLD (GETR WH))
        (SETR WH NIL)
        (TO R/WH))
   (PUSH NP/ (WRD WHOSE)

          (* 'WHOSE' MEANS THAT THE WH-NP IS A POSSESSIVE FOR
          AN NP AFTER THE 'WHOSE')

        (SENDR ADJS (BUILDQ ((POSS +))
                            WH))
        (SETR WH *)
        (TO R/WH))
   (CAT PREP T
        (SETR PREP *)
        (TO R/PREP)))

(R/NIL

          (* HERE TO LOOK FOR A REDUCED RELATIVE--WITHOUT A
          RELATIVE PRONOUN. DETERMINE THE TYPE OF SENTENCE,
          DISPOSE OF THE WH-NP PROPERLY, THEN TRANSFER INTO
          THE CORRECT PLACE IN THE S/ GRAMMAR TO ANALYZE THE
          REST OF THE CLAUSE.)
```

```
(CAT V

      (* A PRESENT PARTICPLE MEANS THE RELATIVE CLAUSE IS
      A PROGRESSIVE SENTENCE WITH THE WH-NP AS SUBJECT;
      A PAST PARTICIPLE MEANS THE RELATIVE CLAUSE IS
      PASSIVIZED, AND THE WH-NP IS HELD AS THE SUBJECT OF
      A PASSIVE SENTENCE USUALLY IS AT STATE VP/V.)


    T
    (COND
      ((AND (GETF PASTPART)
             (VPASSIVE *))
       (HOLD (GETR WH))
       (SETR SUBJ (BUILDQ (NP (PRO SOMETHING))))
       (SETR AGFLAG T))
      ((GETF PRESPART)
       (SETR SUBJ (GETR WH))
       (SETRQ ASPECT (PROGRESSIVE)))
      (T (ABORT)))
    (SETR V *)
    (TO VP/V))
(PUSH NP/ T


      (* AN NP HERE IS THE SUBJECT OF THE RELATIVE CLAUSE;
      THE WH-NP IS EITHER THE OBJECT, INDIRECT OBJECT, OR
      PREP OBJECT--HOLD IT UNTIL WE CAN DETERMINE WHICH.)


    (HOLD (GETR WH))
    (SETR SUBJ *)
    (TO S/NP))
(WRD THERE T
    (SETRQ THERE T)
    (SETR SUBJ (GETR WH))
    (TO S/NP))
(CAT ADJ T


      (* POST-NOMINAL ADJECTIVES ARE PROCESSED AS REDUCED
      RELATIVE COPULAR SENTENCES.
      'INFORMATION AVAILABLE' IS ANALYZED AS 'INFORMATION
      WHICH IS AVAILABLE')


    (SETR SUBJ (GETR WH))
    (SETRQ V BE)
    (SETR OBJ (BUILDQ (@ (ADJ)
                         #                    (*)
                         )
                      FEATURES))
    (TO VP/NP))
(CAT ADV T
    (ADDL VMODS (BUILDQ (ADV *)))
    (TO R/NIL)))
```

```
(R/PREP

          (* LOOKING AT RELATIVE PRONOUN AFTER THE PREP.
          IF THE RELATIVE PRONOUN IS 'WHICH' OR 'WHAT, THE
          WH-NP IS THE OBJECT OF THE PREP.
          FOR 'WHOSE', THE PREP OBJECT IS A NP BEGINNING WITH
          'WHOSE' AND HAVING THE WH-NP AS A POSSESSIVE
          MODIFIER. IN EITHER CASE, A PP IS BUILT AND ADDED TO
          THE VERB MODIFIERS OF THE RELATIVE CLAUSE.
          THIS STATE ALSO GETS THE STRING IN PHASE WITH PATHS
          THAT WENT DIRECTLY FROM R/ TO R/WH.)


   (MEM (WHICH WHOM)
        T
        (ADDL VMODS (BUILDQ (PP (PREP +)
                                +)
                            PREP WH))
        (SETR WH NIL)
        (TO R/WH))
   (PUSH NP/ (WRD WHOSE)
        (SENDR ADJS (BUILDQ ((POSS +))
                            WH))
        (ADDL VMODS (BUILDQ (PP (PREP +)
                                *)
                            PREP))
        (SETR WH NIL)
        (TO R/WH)))


(R/WH

          (* HERE AFTER THE FIRST ATTEMPT AT DECIDING WHAT TO
          DO WITH THE WH-NP. WE ANALYZE OPTIONAL VERB
          MODIFIERS (ADVERBS AND PREP-PHRASES), AND THEN WE
          LOOK FOR AN NP OR 'THERE' IN SUBJECT POSITION.
          IF WE DON'T FIND A VIABLE SUBJECT HERE IN THE STRING
          AND IF THE REGISTER WH STILL CONTAINS THE WH-NP,
          THEN THE WH-NP IS THE SUBJECT.
          OTHERWISE, WE HOLD THE WH-NP FOR LATER.
          IN ANY CASE, WE CUT INTO THE S/ NETWORK AT STATE
          S/NP.)
```

```
    (CAT ADV T
        (ADDL VMODS (BUILDQ (ADV *)))
        (TO R/WH))
    (PUSH PP/ (CAT PREP)
        (ADDL VMODS *)
        (TO R/WH))
    (PUSH NP/ T
        (COND
          ((GETR WH)
            (HOLD (GETR WH))))
        (SETR SUBJ *)
        (TO S/NP))
    (WRD THERE T
        (SETR THERE T)
        (SETR SUBJ (GETR WH))
        (TO S/NP))
    (JUMP S/NP (AND (GETR WH)
                    (CAT V))
        (SETR SUBJ (GETR WH))))


(S/

        (* THIS IS THE INITIAL STATE OF THE WHOLE GRAMMAR.
        BASICALLY, THIS STATE TRIES TO DECIDE WHAT TYPE OF
        SENTENCE WE HAVE: QUESTION, INTERROGATIVE, OR
        IMPERATIVE. THE ARCS SET THE TYPE REGISTER AND
        TRANSFER TO STATES DESIGNED TO HANDLE THE DIFFERENT
        CONSTRUCTIONS. CERTAIN VERB MODIFIERS MAY OPTIONALLY
        PRECEDE THE MAIN BODY OF THE SENTENCE AND PARTITIVE
        CONSTRUCTIONS MAY HAVE BEEN FRONTED FROM A NOUN
        PHRASE; THESE CONSTITUENTS ARE ANALYZED BY LOOPING
        THROUGH S/.)


    (JUMP S/Q (QSTART)

        (* QSTART IS TRUE FOR THE SMALL SET OF WORDS THAT
        CAN START QUESTIONS.)


        (SETRQ TYPE Q))
    (WRD PLEASE (NULL STACK)
                                        (* AT THE TOP LEVEL,
                                        'PLEASE' USUALLY
                                        SIGNIFIES THE BEGINNING
                                        OF AN IMPERATIVE.)
        (ADDL VMODS (BUILDQ (ADV PLEASE)))
        (TO S/IMP))
    (JUMP S/DCL (NOT (QSTART))

        (* THE BEST TEST FOR THE BEGINNING OF A DECLARATIVE
        IS THAT IT NOT BE THE BEGINNING OF A QUESTION.)
```

```
        (SETRQ TYPE DCL))
    (JUMP S/IMP

          (* AN IMPERATIVE CAN OCCUR ONLY AT THE TOP LEVEL;
          IT USUALLY BEGINS WITH AN UNTENSED VERB: 'GIVE ME
          ...')

        (AND (CHECKF V UNTENSED)
             (NULL STACK)))
    (CAT ADV (RPEAT NEGADV)

          (* A NEGATIVE ADVERB ('HARDLY', 'BARELY') USUALLY
          INVOLVES SUBJECT-VERB INVERSION WHEN IT OCCURS AT
          THE BEGINNING OF A SENTENCE.
          THE TYPE IS STILL 'DCL', BUT WE GO TO STATE S/NP TO
          PICK UP THE VERB FOLLOWING THE ADVERB
          ('BARELY HAD HE LEFT ...'). SUBSEQUENT ANALYSIS
          RESEMBLES THE PROCESSING OF YES-NO QUESTIONS.)

        (ADDL VMODS (BUILDQ (ADV *)))
        (SETRQ TYPE DCL)
        (TO S/NP))
    (CAT ADV T
        (ADDL VMODS (BUILDQ (ADV *)))
        (TO S/))
    (PUSH PP/ (WRD OF)

          (* A PARTITIVE EXPRESSION MAY HAVE BEEN FRONTED
          ('OF THE MEN HOW MANY ...'). ANALYZE IT HERE, BUT
          HOLD IT (WITH THE FEATURE 'PARTITIVE') TO BE PICKED
          UP AT STATE NP/QUANT IN THE FIRST NP.)

        (HOLD * (QUOTE ((PARTITIVE))))
        (TO S/))
    (PUSH PP/ (CAT PREP)
        (ADDL VMODS *)
        (TO S/)))

(S/;

          (* A ';' WAS FOUND AT THE END OF THE TOP-LEVEL S,
          INDICATING A SEQUENCE OF SEMICOLON CONJOINED
          SENTENCES, WITH REAL CONJUNCTIONS POSSIBLY FOLLOWING
          THE SEMICOLONS. WE PICK UP THE CONJS IF PRESENT, AND
          PUSH FOR THE FOLLOWING S. WE KEEP THE SEQUENCE OF
          S'S IN SBODY. NOTE: THE STRATEGY HERE IS ALSO USED
          FOR SEMICOLON CONJOINED NPU'S.)
```

```
   (CAT CONJ (OR (NULLR CONJ)
                 (EQ * (GETR CONJ)))

         (* IF WE FIND A CONJ, EITHER IT MUST BE THE FIRST
         ONE ENCOUNTERED CONJOINING THE S'S, OR IT MUST BE
         THE SAME AS PREVIOUSLY ENCOUNTERED ONES.
         THUS WE ACCEPT 'S; AND S; AND S' AND 'S;
         S; AND S', BUT NOT 'S; OR S;
         AND S'.)

       (SETR CONJ *)
       (TO S/;))
   (PUSH S/ T
       (ADDL SBODY *)
       (TO S/;S)))

(S/;S

         (* HERE AFTER PROCESSING ONE S IN A SERIES OF
         SEMICOLON CONJOINED S'S. IF THE CURRENT WORD IS ';',
         THEN ANOTHER S OR CONJ FOLLOWS--GO TO STATE S/;.
         OTHERWISE, POP A COORDINATE LIST OF THE IDENTIFIED
         S'S. NOTE: IF THE END OF THE SERIES IS REACHED WITH
         NO CONJ, THE DEFAULT CONJ IS 'OR'.)

   (WRD ; T
       (TO S/;)
   (POP (BUILDQ (@ (S #)
                   #)
                (COND
                  ((GETR CONJ))
                  (T (QUOTE OR)))
                (REVERSE (GETR SBODY)))
        T))

(S/AUX

         (* HERE AFTER FINDING THE FIRST VERB, WHICH MIGHT
         HAVE BEEN THE MAIN VERB OR AN AUXILIARY.
         LOOP FOR AN OPTIONAL NEG ('NOT') AND UNDO
         'DO-SUPPORT' IF NECESSARY. THEN IF WE ALREADY HAVE
         THE SUBJECT, GO TO VP/V IF IT AGREES WITH THE VERB;
         IF WE HAVEN'T IDENTIFIED THE SUBJECT
         (BECAUSE OF SUBJECT-VERB INVERSION) GO TO S/NO-SUBJ
         TO FIND ONE.)
```

```
    (CAT NEG (NULL NEG)
        (COND
          ((WRD DO MODAL)
            (SETR MODAL NIL)))
        (SETRQ NEG NEG)
        (TO S/AUX))
    (JUMP VP/V (OR (AND (GETR SUBJ)
                        (PNCHECK (GETR SUBJ)
                                 (GETR PNCODE)))
                   (GETR THERE)))
    (JUMP S/NO-SUBJ (NOR (GETR SUBJ)
                         (GETR THERE))))


(S/DCL

          (* WE THINK THIS IS A DECLARATIVE SENTENCE.
          IT MUST BEGIN WITH A SUBJECT NP, A SUBJECT
          COMPLEMENT ('TO HAVE THE INFORMATION IS IMPORTANT'),
          OR 'THERE' IF THERE-INSERTION HAS OCCURRED
          (WE MUST LATER FIND 'BE' OR 'EXIST'))


    (WRD THERE T
        (SETR THERE T)
        (TO S/NP))
    (PUSH NP/ T
        (SETR SUBJ *)
        (TO S/NP))
    (PUSH COMPL/ (OR (WRD (FOR TO THAT))
                     (AND (WRD TO (NEXTWRD))
                          (CAT NEG)))


          (* THERE ARE 4 TYPES OF SUBJECT COMPLEMENTS: 'FOR ME
          TO GO...', 'THAT I WENT ...', 'TO GO ...', 'NOT TO
          GO...'. THE PUSH IS PERMITTED ONLY IF WE HAVE AN
          APPROPRIATE COMPLEMENTIZER.)


        (SETR SUBJ *)
        (TO S/NP)))


(S/IMP

          (* WE RECOGNIZE AN IMPERATIVE SENTENCE, AND SET UP
          REGISTERS ACCORDINGLY. THE SUBJECT IS 'YOU' AND THE
          TENSE IS 'PRESENT'. WE SET THE V AND GO TO VP/HEAD
          TO PICK UP POST-VERB CONSTITUENTS.)
```

```
   (CAT V (GETF UNTENSED)
       (SETRO TYPE IMP)
       (SETR SUBJ (BUILDQ (NP (PRO YOU))))
       (SETR V *)
       (SETR HEAD *)
       (SETRO TNS PRESENT)
       (TO VP/HEAD)))

(S/NO-SUBJ

        (* THERE WAS NO IDENTIFIABLE SUBJECT BEFORE THE
        FIRST VERB. THE SUBJECT MIGHT BE HERE IN THE STRING
        IF S-V INVERSION OCCURRED, OR IT MIGHT BE IN THE WHQ
        REGISTER.)

   (WRD THERE (OR (NULLR WHQ)
                  (PNCHECK (GETR WHQ)
                           (GETR PNCODE)))

        (* ANYTHING IN WHQ MUST AGREE WITH THE VERB AND
        BECOMES THE SUBJECT ('HOW MANY MEN WERE THERE...').
        IF WHQ IS EMPTY ('WERE THERE MANY MEN...') WE MOVE
        ON TO S/THERE.)

       (COND
         ((GETR WHQ)
           (SETR SUBJ (GETR WHQ))))
       (SETR THERE T)
       (TO S/THERE))
   (PUSH NP/ T

        (* WE LOOK FOR AN NP IN THIS POSITION.
        IF NPFEATURES WAS SET (IN THE PUSH FROM STATE
        S/CORE) WE PRESERVE THE OLD VALUE BECAUSE THE
        REGISTER WILL BE RESET BY THIS PUSH
        (AT STATE NP/HEAD). IF THIS PUSH IS SUCCESSFUL, THE
        RESULTING NP MUST AGREE WITH THE VERB AND BECOMES
        THE SUBJECT. OUR INDECISION ABOUT THE WHQ IS
        RESOLVED--IT CANNOT BE THE SUBJECT SO IT IS HELD TO
        BE PICKED UP AS AN OBJECT OR PREP OBJECT.
        WE ALSO HOLD THE NPFEATURES ASSOCIATED WITH IT, FOR
        LATER RESUMPTION. FINALLY, THE DO-SUPPORT NECESSARY
        FOR S-V INVERSION IS UNDONE.)
```

```
        (! (SETR HOLDNPFEATURES (GETR NPFEATURES)))
        (COND
          ((NULL (PNCHECK * (GETR PNCODE)))
           (ABORT)))
        (SETR SUBJ *)
        (COND
          ((GETR WHQ)
           (HOLD (GETR WHQ)
                 (GETR HOLDNPFEATURES))))
        (COND
          ((WRD DO MODAL)
           (SETR MODAL NIL)))
        (TO VP/V))
    (JUMP VP/V

          (* IF 'THERE' AND NP DIDN'T WORK BUT WE HAVE A WHQ
          WHICH AGREES WITH THE VERB, WE TRY THAT AS THE
          SUBJECT.)


        (AND (GETR WHQ)
             (PNCHECK (GETR WHQ)
                      (GETR PNCODE)))
        (SETR SUBJ (GETR WHQ))))


(S/NP

          (* HERE AFTER OUR FIRST ATTEMP AT FINDING A NP,
          EITHER AS A DECLARATIVE SUBJECT OR AS A QUESTION
          WORD (WHO).)


   (POP

          (* IF WE REACHED THE END OF THE STRING AT THE TOP
          LEVEL, WE BUILD A NOUN-PHRASE UTTERANCE AS THE PARSE
          OF THE SENTENCE ('INFORMATION ON ...' OR 'WHICH
          MAN').)


        (BUILDQ (S NPU *)
                (COND
                  ((GETP SUBJ))
                  ((GETR WHQ))))
        (POP STRING STACK (GETR VMODS)))
    (CAT V (GETF INS)
```

```
        (* USUALLY WE FIND A TENSED VERB AT THIS STRING
        POSITION, EITHER AS THE FIRST WORD IN A QUESTION
        (IF WE JUMPED FROM S/Q), OR FOLLOWING AN NP OR
        'THERE'. IF IT IS A MODAL ('WOULD', 'COULD', ETC.)
        WE PUT IT IN THE MODAL REGISTER, OTHERWISE IN V.
        IF WE ARE IN A WH-QUESTION AND THE VERB WAS NOT AN
        AUXILIARY ('WHO HIT JOHN') THEN THE WHQ IS THE
        SUBJECT. IN ANY CASE, SAVE THE TENSE AND
        PERSON-NUMBER CODE.)


      (COND
        ((MODAL)
          (SETR MODAL *))
        (T (SETR V *)))
      (COND
        ((AND (GETP WHQ)
              (NOR (MODAL)
                   (WRD (HAVE BE))))
          (SETR SUBJ (GETR WHQ))
          (SETR WHQ NIL)))
      (SETR TNS (GETF TNS))
      (SETR PNCODE (GETF PNCODE))
      (TO S/AUX))
   (WRD ; (NULL STACK)

        (* AT THE TOP LEVEL, A SEMICOLON HERE INTRODUCES A
        SEQUENCE OF NP'S TO BE PARSED AS CONJOINED NPU'S.
        THE GRAMMAR FOR THIS BEGINS AT NPU/;
)


      (ADDL NPU (GETR SUBJ))
      (TO NPU/;))
   (JUMP S/DCL (WPD IDQ TYPE)

        (* WE ARE IN AN INDIRECT QUESTION, PUSHED TO FROM
        VP/HEAD ('I KNOW WHO YOU ARE') THE WHQ REGISTER IS
        SET, BUT INVERSION HASN'T OCCURRED, I.E., THE
        BEGINNING OF THE CLAUSE AFTER THE Q-WORD LOOKS LIKE
        A DECLARATIVE. ERGO, JUMP TO S/DCL.)


      (HOLD (GETR WHQ)
            (GETR NPFEATURES))
      (SETR WHQ NIL)))
```

```
(S/Q

        (* THE SENTENCE BEGINS LIKE A QUESTION.
        THREE POSSIBILITIES: (1) A QWORD--A WH WORD THAT
        FUNCTIONS AS A PRONOUN ('WHO','WHAT');
        (2) AN AUXILIARY VERB; (3) A Q-DETERMINER
        ('WHICH MAN'))

   (CAT QWORD T

        (* THE DICTIONARY ENTRIES FOR QWORDS ARE COMPLETE NP
        OR ADVERBIAL STRUCTURES. WE COPY THEM SO WE DON'T
        DESTROY THE DICTIONARY ENTRIES BY FUTURE OPERATIONS.
        THE FEATURE 'SUBJ/OBJ' INDICATES THAT THE QWORD CAN
        REPRESENT EITHER THE SUBJECT OR THE OBJECT, SO WE
        STORE IT IN WHO UNTIL FURTHER INFORMATION ENABLES US
        TO DECIDE. IF THE QWORD LACKS THIS FEATURE, THEN IT
        CANNOT BE THE SUBJECT ('WHOM'); WE HOLD IT FOR
        POST-VERBAL PROCESSING.)

     (COND
       ((GETF SUBJ/OBJ)
         (SETR WHQ (COPY *)))
       (T (HOLD (COPY *))))
     (TO S/NP))
   (JUMP S/NP (CAT V))
   (CAT QDET T

        (* CURRENTLY, THE NP/ LEVEL DOES NOT PROCESS
        QUESTION DETERMINERS. WE PICK THEM UP HERE AND PUSH
        INTO THE MIDDLE OF THE NP/ NETWORK FROM S/QDET,
        SENDING THE QDET DOWN.)

      (SETR DET *)
      (TO S/QDET)))

(S/QDET

        (* DET CONTAINS THE Q-DETERMINER FOUND AT STATE S/Q.
        WE PUSH INTO THE MIDDLE OF THE NP/ NETWORK TO FIND
        THE WHO NOUN-PHRASE. WE INITIALIZE THE REGISTER QDET
        TO PREVENT REDUCED RELATIVE CLAUSES WITHIN THE NP:
        'THE MAN I SAW...' AND 'HOW MANY MEN WHO I SAW...'
        ARE ALLOWED, BUT 'HOW MANY MEN I SAW...' IS OUT.)
```

```
   (PUSH NP/ART T
       (SENDR DET (GETR DET))
       (SENDRQ QDET T)
       (SETR WHQ *)
       (TO S/NP)))

(S/SADV
   (PUSH NP/ T
       (ADDL VMODS (BUILDQ (ADVP (ADV +)
                                 *)
                           SADV))
       (TO S/VP)))

(S/THERE

          (* THIS STATE IS PLACED BETWEEN S/NO-SUBJ AND VP/V
          TO ALLOW FOR EXTRAPOSED RELATIVE CLAUSES WITH THERE
          INSERTION IN QUESTIONS: 'HOW MANY MEN WERE THERE WHO
          BOUGHT ...'. THE RESUME ACTION WILL MOVE THE
          RELATIVE TO ITS PROPER LOCATION IN THE NP, WHICH
          THEN BECOMES THE SUBJECT. NOTICE THAT UNLESS
          NPFEATURES HAS BEEN SET, THIS ARC IS ESSENTIALLY A
          NO-OP. THUS, SINCE QWORDS DON'T SET NPFEATURES
          (QDETS DO), 'WHO WAS THERE WHO DID ...' IS
          (PERHAPS ERRONEOUSLY) NOT ALLOWED.)

   (DO (SETQ FEATURES (GETR NPFEATURES))
       (RESUME)
       (COND
         ((GETR NPFEATURES)
          (SETR SUBJ *)))
       (JUMP VP/V)))

(S/VP

          (* HERE WHEN THE VERB-PHRASE OF THIS S HAS BEEN
          NEARLY COMPLETED. THERE MIGHT BE SOME ADVERBS OR
          PP'S STILL ON THE HOLD LIST, WHICH WE PICK UP HERE.
          ALSO, AT THE TOP LEVEL, THERE MIGHT BE SOME TERMINAL
          PUNCTUATION, OR A SEMICOLON, INDICATING THAT THIS IS
          THE FIRST ITEM IN A SERIES OF CONJOINED S'S.
          THE USUAL CASE, HOWEVER, IS TO POP THE ANALYZED S
          STRUCTURE.)
```

```
    (WRD , T
        (TO S/VP))
    (CAT ADV (RPEAT TRANSADV)
        (SETR SADV *)
        (TO S/SADV))
    (VIP PP T
        (ADDL VMODS *)
        (TO S/VP))
    (VIR ADV T
        (ADDL VMODS *)
        (TO S/VP))
    (MEM (%. ? !)
        (NULL STACK)

           (* NOTE THAT A TERMINATING QUESTION MARK OVERRIDES
           THE SYNTACTIC TYPE OF THE SENTENCE: 'I NEED SOME
           INFORMATION?' IS A QUESTION, NOT A DECLARATIVE.)

        (COND
          ((AND (WRD ?)
                (NOT (WRD Q.TYPE)))
           (SETRQ TYPE Q)))
        (TO S/VP))
    (WRD ; (NULL STACK)
        (ADDL SBODY (SBUILD))
        (TO S/;))
    (POP (SBUILD)
        T))

(VP/AGT

           (* HERE IF THE SENTENCE IS PASSIVE, WE HAVE NOT YET
           FOUND THE AGENT, BUT WE HAVE SEEN THE PREPOSITION
           BY, WHICH MIGHT INTRODUCE THE AGENT NP.
           USALLY, THE AGENT NP WOULD BE IDENTIFIED HERE 'BY' A
           PUSH, BUT IN A QUESTION OR RELATIVE CLAUSE, THE
           AGENT MIGHT HAVE BEEN FRONTED, LEAVING THE 'BY'
           DANGLING: 'WHO IS THE INFORMATION NEEDED BY' OR 'THE
           MAN THE INFORMATION IS NEEDED BY...'.
           IN THESE CASES, THE NP HAS BEEN HELD, AND ARC 2
           PICKS IT UP. THE RESUME ACTION ALLOWS FOR AN
           EXTRAPOSED RELATIVE CLAUSE OR PP.)
```

```
   (PUSH NP/ T
       (SETR SUBJ *)
       (SETR AGFLAG NIL)
       (TO VP/VP))
   (VIR NP T
       (RESUME)
       (SETR SUBJ *)
       (SETR AGFLAG NIL)
       (TO VP/VP)))


(VP/HEAD

          (* HERE WHEN WE HAVE MADE FIRM DECISIONS ABOUT THE
          MAIN VERB AND THE SUBECT. WE LOOK FOR POST-VERBAL
          MODIFIERS (OBJECTS, SENTENTIAL COMPLEMENTS,
          PARTICLES, PREDICATE ADJECTIVES), MANY OF WHICH ARE
          SPECIFIED BY ROOT FEATURES ON THE VERB.)


   (JUMP VP/NP (RFEAT INTRANS V)

          (* IF THE MAIN VERB IS MARKED INTRANSITIVE, THERE IS
          NO PREDICATE COMPLEMENT DIRECTLY TIED TO THE
          VERB--WE SKIP TO VP/NP.)

)
   (PUSH S/Q (AND (VTRANS V)
                  (WRD (WHICH WHO WHAT WHOSE)))

          (* CERTAIN VERBS CAN TAKE INDIRECT QUESTIONS AS
          THEIR OBJECTS (E.G. 'KNOW'). THESE ARE MARKED AS
          ORDINARY TRANSITIVES IN THE DICTIONARY, SO IN ORDER
          TO RECOGNIZE THESE CONSTRUCTIONS, WE ALLOW THE
          POSSIBILITY THAT ALL TRANSITIVE VERBS CAN TAKE THESE
          OBJECTS. ('I KNOW WHO WANTED THE INFORMATION.'))

       (SENDRQ TYPE IDQ)
       (SETR OBJ (BUILDQ (NP *)))
       (TO VP/NP))
   (PUSH NP/ (AND (VTRANS V)
                  (NOT (AND (WRD BE V)
                            (GETR WHQ))))

          (* HERE WE PICK UP THE REGULAR OBJECT OF TRANSITIVE
          VERBS. NOTE THAT FOR A WHO-QUESTION WITH 'BE' AS THE
          MAIN VERB ('WHO IS THE LEADER?'), THE SUBJECT
          ('THE LEADER') WAS PICKED UP AT STATE S-NO-SUBJ, AT
          WHICH POINT THE WHO WAS HELD.
          THUS WE DON'T LOOK FOR THE OBJECT ON THIS ARC, BUT
          RATHER ON THE SUBSEQUENT VIR ARC.)
```

```
                (SETR OBJ *)
                (T  VP/NP))
            (CAT ADJ (RFEAT COPULA V)

(                (* HERE WE PICK UP PREDICATE ADJECTIVES OF COPULAR
                 VERBS ('BE', 'BECOME', 'SEEM'))

                (SETR OBJ (BUILDQ (@ (ADJ)
                                     #                    (*)
                                     )
                                  FEATURES))
                (T  VP/NP))
            (VIB NP (VTRANS V)

                 (* FOR RELATIVE CLAUSES, PASSIVES, AND
                 WHQ-BE-QUESTIONS, THE DIRECT OBJECT HAS BEEN HELD;
                 WE PICK UP HERE, LOOKING FOR POSSIBLE EXTRAPOSED
                 RELATIVES.)

( JS
                (RESUME)
                (SETR OBJ *)
                (T  VP/NP))
            (PUSH COMPL/ (AND (WRD THAT)
                              (RFEAT THATCOMP V))

                 (* VERBS MARKED 'THATCOMP' CAN TAKE A THAT-CLAUSE AS
                 A COMPLEMENT ('I BELIEVE THAT THEY...').)

                (SETR COMPL *)
(VP/            (T  VP/NP))
            (PUSH COMPL/ (AND (WRD (FOR TO))
                              (RFEAT FORCOMP V))

(
                 (* 'FORCOMP' VERBS TAKE A FOR- OR TO-COMPLEMENT: 'WE
                 HOPE FOR JOHN TO COME', 'WE WANT TO COME'.
                 FOR A TO-COMPLEMENT, THE SUBJECT OF THE COMPLEMENT
                 IS THE SAME AS THE SUBJECT OF THE SENTENCE.)
```

```
        (OR (AND (WRD (FOR THAT))
                 (EQUAL (GETR SUBJ)
                        (QUOTE (NP (PRO IT)))))
            (AND (WRD THAT)
                 (GETR AGFLAG)))
        (COND
          ((GETR AGFLAG)
           (SETRQ AGFLAG NIL)))
        (SETR SUBJ *)
        (TO VP/VP))
    (PUSH FOR/NP (AND (RFEAT TOCOMP V)
                      (GETR OBJ))

           (* A TO-COMPLEMENT CAN OCCUR AFTER THE OBJECT: 'I
           PROMISED JOHN TO GO' OR 'I WANTED JOHN TO GO'.
           FOR MOST VERBS, THE SUBJECT OF THE COMPLEMENT IS THE
           OBJECT ('JOHN') OF THE MAIN SENTENCE, BUT VERBS
           MARKED 'SUBJLOW' HAVE THE SUBJECT OF THE MAIN
           SENTENCE PASSED DOWN (E.G. 'PROMISE'). FOR
           'TRANSCOMP' VERBS, THE OBJECT PASSED DOWN TO BE
           SUBJECT REMAINS AS THE TOP-LEVEL OBJECT
           ('PERSUADE'), BUT THIS IS NOT ALWAYS TRUE
           ('EXPECT'))

        (SENDR SUBJ (COND
                    ((RFEAT SUBJLOW V)
                      (GETR SUBJ))
                    (T (GETR OBJ))))
        (COND
          ((NOT (RFEAT TRANSCOMP V))
           (SETR OBJ NIL)))
        (SETR COMPL *)
        (TO VP/VP))
    (CAT PREP (SETQ TEMP (VPARTICLE V))
                                        (* A PARTICLE CAN OCCUR
                                        AFTER THE OBJECT: 'LOOK
                                        THE INFORMATION UP')
        (SETR V TEMP)
        (SETR HEAD TEMP)
        (TO VP/VP))
    (PUSH NP/ (AND (RFEAT INDOBJ V)
                   (GETR OBJ))

           (* A NP CAN OCCUR IN THIS POSITION IF THE VERB CAN
           TAKE AN INDIRECT OBJECT, WHAT WE THOUGHT WAS THE
           OBJECT WAS REALLY THE INDIRECT OBJECT, AND THE NP
           HERE IS TO BE THE DIRECT OBJECT.
           ('I GAVE JOHN THE INFORMATION' --> 'I GAVE THE
           INFORMATION TO JOHN'))
```

```
        (ADDL VMODS (BUILDQ (PP (PREP TO)
                                +)
                            OBJ))
        (SETR OBJ *)
        (TO VP/VP))
    (PUSH COMPL/ (AND (WRD (FOR THAT))
                      (RFEAT INDOBJ V))

          (* THE DIRECT OBJECT CAN ALSO BE A COMPLEMENT, FOR
          CERTAIN VERBS THAT ALLOW INDIRECT OBJECTS: 'I TOLD
          MARY THAT...')

        (ADDL VMODS (BUILDQ (PP (PREP TO)
                                +)
                            OBJ))
        (SETR OBJ NIL)
        (SETR COMPL *)
        (TO VP/VP))
    (JUMP VP/VP T
                                                (* FINALLY, JUMP TO
                                                VP/VP)))

(VP/V

          (* THE FIRST VERB CAN BE FOLLOWED BY OTHER VERBS TO
          FILL OUT THE PERFECT-PROGRESSIVE-PASSIVE AUXILIARY
          STRUCTURE, ADVERBS AND THE SUBJECT OF A
          THERE-INSERTED SENTENCE CAN BE INTERSPERSED BETWEEN
          THE VERBS--WE LOOP FOR THEM HERE.)

    (CAT V T

          (* VERBS AFTER THE MAIN VERB MUST BE PARTICIPLES OR
          UNTENSED FORMS. IF A PAST PARTICIPLE, THE PREVIOUS
          VERB IN THE SEQUENCE MUST BE EITHER 'HAVE'
          (ASPECT=PERFECT) OR 'BE' (SENTENCE IS PASSIVE, IF
          POSSIBLE). A PRESENT PARTICIPLE MUST BE PRECEDED BY
          'BE' (ASPECT=PROGRESSIVE). OTHERWISE, THE CURRENT
          WORD MUST BE AN UNTENSED VERB AND THE REGISTER V
          MUST BE EMPTY (BECAUSE THE FIRST VERB WAS A MODAL).
          IF ANY OF THESE CONDITIONS IS SATISFIED, WE REPLACE
          THE VERB BY THE ROOT FORM OF THE CURRENT WORD.)
```

```
      (COND
        ((GETF PASTPART)
          (COND
            ((AND (WRD BE V)
                  (VPASSIVE *))
              (HOLD (GETR SUBJ)
                    (GETR NPFEATURES))
              (SETR SUBJ (BUILDQ (NP (PRO SOMETHING))))
              (SETR AGFLAG T))
            ((AND (NULLR ASPECT)
                  (WRD HAVE V))
              (SETRQ ASPECT (PERFECT)))
            (T (ABORT))))
        ((GETF PRESPART)
          (COND
            ((WRD BE V)
              (ADDR ASPECT (QUOTE PROGRESSIVE)))
            ((WRD POSS-ING TYPE))
            (T (ABORT))))
        ((OR (NOT (GETF UNTENSED))
             (GETR V))
          (ABORT)))
      (SETR V *)
      (TO VP/V))
  (PUSH NP/ (AND (GETR THERE)
                 (NULLR SUBJ)
                 (WRD (BE EXIST)
                      V))
                                          (* HERE WE PUSH FOR THE
                                          SUBJECT OF A
                                          THERE-INSERTED
                                          SENTENCE.)
      (COND
        ((NOT (PNCHECK * (GETR PNCODE)))
          (ABORT)))
      (SETR SUBJ *)
      (TO VP/V))
  (JUMP VP/HEAD (GETR SUBJ)
```

(* IF WE HAVE THE SUBJECT, WE CAN ASSUME THAT WE ALSO HAVE THE MAIN VERB (USUALLY, THIS WILL BE TRUE BECAUSE WE WOULD HAVE LOOPED THROUGH THE FIRST ARC AS LONG AS POSSIBLE) AND JUMP TO VP/HEAD TO LOOK FOR POST-VERB CONSTITUENTS. IF THE V REGISTER IS EMPTY (THE FIRST AND ONLY VERB WAS A MODAL), WE ABORT UNLESS THE MODAL WAS 'DO'--WE ALLOW 'DO' TO BECOME THE MAIN VERB.)

```
        (COND
          ((NULLR V)
            (COND
              ((WRD DO MODAL)
                (SETRQ V DO)
                (SETR MODAL NIL))
              (T (FBCRT))))
          (T (COND
               ((AND (GETR THERE)
                     (WRD BE V))
                (SETRQ V EXIST)))))
        (SETR HEAD (GETR V)))
    (CAT ADV T
        (ADDL VMODS (BUILDQ (ADV *)))
        (TO VP/V)))


(VP/VP

          (* THE ELEMENTS OF THE VERB PHRASE WHICH ARE CLOSELY
          TIED TO THE MAIN VERB (E.G. COMPLEMENTS) HAVE BEEN
          PROCESSED. VARIOUS ADDITIONAL MODIFIERS ARE STILL
          PERMITTED (ADVERBS, PREP-PHRASES). ALSO, WE LOOK FOR
          THE AGENT OF PASSIVE SENTENCES AND THE OBJECT OR
          SUBJECT OF POSS-ING COMPLEMENTS, IF THESE HAVE NOT
          BEEN ALREADY IDENTIFIED.)


    (WRD BY (GETR AGFLAG)

          (* AGFLAG IS SET IF WE HAVEN'T FOUND THE SUBJECT OF
          A PASSIVE SENTENCE. 'BY' CAN INTRODUCE IT.)


        (TO VP/AGT))
    (WRD BY (AND (WRD POSS-ING TYPE)
                 (NULLR OBJ)
                 (NULLR SUBFLAG))

          (* IN A POSS-ING COMPLEMENT WHERE THE SUBJECT WAS
          SENT DOWN FROM THE POSSESSIVE AT STATE NP/DET, THE
          SENT SUBJECT MIGHT REALLY BE THE OBJECT IF NO OBJECT
          WAS FOUND ('THE DUCK'S SHOOTING BY THE HUNTERS ...')
          AND THE REAL SUBJECT CAN FOLLOW A 'BY' IN THIS
          POSITION.)
```

```
        (SETR OBJ (GETR SUBJ))
        (TO ING/BY))
    (MEM (OF BY)
        (GETR SUBFLAG)

          (* IN A POSS-ING COMPLEMENT, IF THE SUBJECT WAS NOT
          SENT DOWN FROM THE POSSESSIVE, IT CAN APPEAR
          FOLLOWING EITHER 'OF' OR 'BY': 'THE SHOOTING OF THE
          HUNTERS ...' OR 'THE SHOOTING BY THE HUNTERS')

        (SETR SUBFLAG NIL)
        (TO ING/BY))
    (CAT ADV T
        (ADDL VMODS (BUILDQ (ADV *)))
        (TO VP/VP))
    (PUSH PP/ (CAT PREP)
        (ADDL VMODS *)
        (TO VP/VP))
    (JUMP S/VP T))
)
STOP
```

References

Bobrow, D.G., Murphy,D.P., and Teitelman, W.,"The BBN-LISP System", BBN Report 1677, Bolt Beranek and Newman Inc., Cambridge, Mass., April, 1968.

Chomsky, N., Aspects of the Theory of Syntax. Cambridge: M.I.T. Press, 1965.

Chomsky, N., Syntactic Structures. The Hauge: Mouton and Co., 1957.

Defense Documentation Center, DDC Remote On-Line Retrieval System Operator's Manual, DSA DDCM 4185.3 (Provisional), Defense Documentation Center, Alexandria, Virginia, May, 1970.

Myer, T.R. and Barnaby, J.R., TENEX Executive Language, Bolt, Beranek and Newman Inc., Cambridge, Mass., Jan., 1971.

Norman, E., "LISP Reference Manual for the UNIVAC 1108", University of Wisconsin Computing Center, Madison, Wisconsin, 1969.

Petrick, S., A Recognition Procedure for Transformational Grammars. Unpublished doctoral dissertation, M.I.T., 1965.

Stockwell, R., Schachter, and Partee, B., Integration of Transformational Theories on English Syntax. UCLA, 1968.

Watt, W.C., "Habitability", American Documentation, Vol. 19, No. 3, July 1968.

Woods, W.A., "Augmented Transition Networks for Natural Language Analysis", Harvard Computation Laboratory Report No. CS-1, Harvard University, Cambridge, Mass., Dec., 1969.

Woods, W.A., "Semantics for a Question-Answering System", Ph.D. thesis, Harvard University, Cambridge, Mass., Aug., 1967.

Woods, W.A., Transition Network Grammars for Natural Language Analysis. Communications of the ACM, 13, 591-602, Oct. 1970.

Woods, W.A., "Procedural Semantics for a Question-Answering Machine," AFIPS Conference Proceedings, Vol. 33 (1968, FJCC). (Enclosed as Appendix A).

Zwicky, A., Friedman, J., Hall, B., and Walker, D., "The MITRE Syntactic Analysis Procedure for Transformational Grammars", Proceedings of the Fall Joint Computer Conference, 1965, 317-326.

Appendix C.

Semantic Rules

```
(PRIN1 (QUOTE "CREATED ") T)
(PRIN1 (QUOTE " 1-JUL-71 14:11:00") T)
(TERPRI T)
  (PRINT (QUOTE DDCRULESCOMS)
         T)
  (RPAQQ DDCRULESCOMS ((R: DDCRULES)
          (F: RULEFNS)
          (V: TREEFRAGS)
          (V: RULELISTS)
          (V: (TOPICRULES))
          (V: MORPHOLOGY)))
(PRINT (QUOTE RULES:) T)
(PRINT (QUOTE DDCRULES) T)
(RPAQQ DDCRULES (D:ALL D:ALL\ONES ANY:TERM D:ANAPHORA DD0 DD1 DD1:5
DD10 DD11 DD12 DD13 DD14 DD15 DD16 DD2 DD2:5 DD3 DD4 DD5 DD6 DD7 DD8
DD9 NP:NPR PR1 PR2 PR3 PR4 PR5 PR6 R:ADJ R:BIBLIOGRAPHY R:DOC-ON R:N-DOC
R:PP R:REL REFRULE REFRULE? S:AND S:BE-INTERESTED S:DCL S:GIVE S:LIKE
S:IMP S:NPU S:OR S:PAPER-HAVE S:SEARCH S:WHQ S:YES/NO SS30 SS31 SS32
SS33 SS34 SS35 SS36 SS37 SS38 SS39 SS40 SS41 TOPIC\ADJ TOPIC\ADJ-N
TOPIC\ADJ-NPR TOPIC\ADJ-NP TOPIC\AND-NP TOPIC\AND-S TOPIC\ESP TOPIC\N
TOPIC\NOM TOPIC\NP-S TOPIC\NR-NP TOPIC\NOT-NP TOPIC\NOT-S TOPIC\NPR
TOPIC\OR-NP TOPIC\OR-S TOPIC\PP TOPIC\REL TOPIC\S-COMPL TOPIC\S-NP
TOPIC\S-OBJ TOPIC\S-PP TOPIC\TERM TOPIC\TERM2 N:DOCUMENT S:CONCERN
S:PERTAIN S:BE-ABOUT R:DOC TOPIC\AUTHOR TOPIC\AUTHOR2 TOPIC\NP-COMPL
TOPIC\PP-COMPL TOPIC\V-INTRANS TOPIC\V-TRANS TOPIC\V-TRANS2
S:RUN-A-SEARCH S:TOPIC TOPIC\EMPHASIS TOPIC\ADJ:SUPER TOPIC\ADJ:COMP))
(DEFINEQ

(D:ALL
    (NP-DET T)
        --  (QUANT (FOR EVERY X / (# 0 NRULES) : (# 0 RRULES) ; DLT))  )

(D:ALL\ONES
    (NP-DET (EQU 1 ALL))
    (NP-PRO (EQU 1 ONES))
    (NP-PP (EQU 1 OF))
        --  (QUOTE (# 3 2 ALL))  )

(ANY:TERM
    (T T)
        --  (LIST (QUOTE QUOTE) (EVAL (CONS (QUOTE NPR) (QUOTE (#
0 TERM))))  )

(D:ANAPHORA
    (OR (NP-PRO (NOT (OR (EQU 1 I)
                         (EQU 1 YOU)
                         (EQU 1 ONES))))
        (NP-DET (OR (EQU 1 THIS)
                    (EQU 1 THAT)
                    (EQU 1 THESE)
                    (EQU 1 THOSE))))
        --  (PROG (TEMP) (COND ((SETQ TEMP (ANTECEDANT (QUOTE (# 0
IDENTITY)))) (SETQ QVAR TEMP) (MAPC (SCOPEVARS TEMP) (FUNCTION ANTEQUANT)
(RETURN (ANTEQUANT TEMP))) (T (RETURN QVAR))))  )
```

```
(DDØ
    ((NP ((NPR NIL 1)))
     T)
        -- (LIST (QUOTE QUOTE) (EVAL (CONS (QUOTE NPR) (QUOTE (#
1 1 TERM))))))  )

(DD1
    (NP←DET (OR (EQU 1 SOME)
                (EQU 1 A)
                (EQU 1 AN)
                (EQU 1 ANY)))
        -- (QUANT (FOR SOME X / (# Ø NRULES) : (# Ø RRULES) ; DLT))  )

(DD1:5
    (NP←DET (EQU 1 NIL))
        -- (QUANT (FOR GEN X / (# Ø NRULES) : (# Ø RRULES) ; DLT))  )

(DD1Ø
    (NP←DET (AND (OR (EQU 1 WHO THING SG MANY)
                     (EQU 1 HOWMANY))
                 (EQU 2 PL)))
        -- (QUANT (PRINTOUT (NUMBER X / (# Ø NRULES) : (AND (# Ø
RRULES) DLT))))  )

(DD11
    (NP←DET←COMP (OR (EQU 1 AT LEAST)
                     (EQU 1 AS MANY AS)
                     (EQU 1 ATLEAST)
                     (EQU 1 ASMANYAS)))
        -- (QUANT (FOR (1 . 2) MANY X / (# Ø NRULES) : (# Ø RRULES)
; DLT))  )

(DD12
    (NP←DET←COMP (EQU 1 EXACTLY))
        -- (QUANT (EQUAL (1 . 2) (NUMBER X / (# Ø NRULES) : (AND
(# Ø RRULES) DLT))))  )

(DD13
    (NP←DET←COMP (OR (EQU 1 MORE THAN)
                     (EQU 1 MORETHAN)))
        -- (QUANT (FOR (GREATER N (1 . 2)) MANY X / (# Ø NRULES)
: (# Ø RRULES) ; DLT))  )

(DD14
    (NP←DET←COMP (OR (EQU 1 FEWER THAN)
                     (EQU 1 LESS THAN)
                     (EQU 1 FEWERTHAN)
                     (EQU 1 LESSTHAN)))
        -- (QUANT (NOT (FOR (1 . 2) MANY X / (# Ø NRULES) : (# Ø
RRULES) ; DLT)))  )
```

```
(DD15
    (NP*DET*COMP (OR (EQU * AT MOST)
                     (EQU * ATMOST)))
        --  (QUANT (NOT (FOR (GREATER N (1 . 2)) MANY X / (# Ø NRULES)
: (# 2 RRULES) ; DLT)))  )

(DD16
    (NP*DET (AND (OR (EQU * THE)
                     (EQU * ALL)
                     (EQU * NIL))
                 (EQU 2 PL)))
        --  (PROG1 (SETQ SEM (SUBLIS (QUOTE ((FOR . UNION) (DLT SETOF
X / (# Ø NRULES) : (# Ø RRULES) ; T))) QUANT)) (SETQQ QUANT DLT))  )

(DD2
    (NP*DET (AND (OR (EQU * EACH)
                     (EQU * EVERY))
                 (EQU 2 SG)))
        --  (QUANT (FOR EVERY X / (# Ø NRULES) : (# Ø RRULES) ; DLT))  )

(DD2:5
    (NP*DET (AND (EQU 1 ALL)
                 (EQU 2 PL)))
        --  (DO (DSUBST (QUOTE EVERY) (QUOTE GEN) QUANT) (QUANT (FOR
EVERY X / (# Ø NRULES) : (# Ø RRULES) ; DLT)))  )

(DD3
    (NP*DET (EQU 1 NEG SOME))
        --  (QUANT (NOT (FOR SOME X / (# Ø NRULES) : (# Ø RRULES)
; DLT)))  )

(DD4
    (NP*DET (OR (AND (EQU * NEG EVERY)
                     (EQU 2 SG))
                (AND (EQU * NEG ALL)
                     (EQU 2 PL))))
        --  (QUANT (NOT (FOR EVERY X / (# Ø NRULES) : (# Ø RRULES)
; DLT)))  )

(DD5
    (NP*DET (AND (OR (EQU * THE)
                     (EQU * THIS)
                     (EQU * THAT))
                 (EQU 2 SG)))
        --  (QUANT (FOR THE X / (# Ø NRULES) : (# Ø RRULES) ; DLT))  )

(DD6
    (NP*DET (AND (OR (EQU * WHO)
                     (EQU * WHICHQ)
                     (EQU * WHICH)
                     (EQU * WHAT))
                 (EQU 2 SG)))
        --  (QUANT (FOR THE X / (# Ø NRULES) : (AND (# Ø RRULES) DLT)
```

```
; (PRINTOUT X)))  )

(DD7
    (NP←DET (AND (EQU 1 THE)
                 (EQU 2 PL)))
        --  (QUANT (FOR EVERY X / (# Ø NRULES) : (# Ø RRULES) ; DLT))  )

(DD8
    (NP←DET (AND (OR (EQU 1 WHICH)
                     (EQU 1 WHAT)
                     (EQU 1 WHQ)
                     (EQU 1 WHICHQ))
                 (EQU 2 PL)))
        --  (DO (DSUBST (QUOTE EVERY) (QUOTE GEN) QUANT) (QUANT (FOR
EVERY X / (# Ø NRULES) : (AND (# Ø RRULES) DLT) ; (PRINTOUT X))))  )

(DD9
    (NP←DET←MANY (T))
        --  (QUANT (FOR (EQ N (1 . 1)) X / (# Ø NRULES) : (# Ø RRULES)
; DLT))  )

(NP:NPR
    (NP←NPR T)
        --  (LIST (QUOTE QUOTE) (EVAL (CONS (QUOTE NPR) (QUOTE (#
1 1 TERM)))))  )

(PR1
    (S←Q-NEG (NOT (LEAFMEMB P
                            (QUOTE (WHQ WHICHQ WHEN WHERE WHY HOW
                                    HOWMANY)))))
        --  (PRED (TEST (# Ø SRULES)))  )

(PR2
    (S←Q (NOT (LEAFMEMB P (QUOTE (WHQ WHICHQ WHEN WHERE WHY HOW HOWMANY)
                                 ))))
        --  (PRED (TEST (# Ø SRULES)))  )

(PR3
    (S←NEG (*))
        --  (PRED (NOT (# Ø SRULES)))  )

(PR4
    (OR ((S (DCL))
         T)
        ((S (REL))
         T)
        ((S (POSS-ING))
         T))
        --  (PRED (# Ø SRULES))  )
```

```
(PR5
    (S←IMP T)
        --  (PRED (DO (# 2 SRULES)))  )

(PR6
    (S←NP T)
    (S←VP T)
        --  (PRED (# 2 SRULES))  )

(R:ADJ
    (NP←N T)
    (NP←ADJ (NOT (EQ (CAR (# 1)
                          (QUOTE NP)))))
        --  (PROG (TEMPRELS) (SETQ TEMPRELS (QUOTE ((S REL (NP (DET
WHI) (N (# 1 1 HEAD)) (NU SG)) (NP (V BE) (# 2 1 IDENTITY)))))) (RELTAG
TEMPRELS) (RETURN (SEM (CAR TEMPRELS))))  )

(R:BIBLIOGRAPHY
    (NP←N (MEM 1 BIBLIOGRAPHY))
    (OR (NP←PP (OR (EQU 1 ON)
                   (EQU 1 ABOUT)
                   (EQU 1 OF)))
        (NP←PP←PP#4 (AND (MEM 4 DOCUMENT)
                         (OR (EQU 1 ON)
                             (EQU 2 ABOUT)))))
        --  (QUOTE (ABOUT X (# 2 2 TOPIC)))  )

(R:DOC-ON
    (NP←N (MEM 1 DOCUMENT))
    (NP←PP (OR (EQU 1 ABOUT)
               (EQU 1 ON)
               (EQU 1 FOR)
               (EQU 1 TO)
               (EQU 1 OF)
               (EQU 1 IN)))
        --  (QUOTE (ABOUT X (# 2 2 TOPIC)))  )

(R:N-DOC
    (NP←N (MEM 1 DOCUMENT))
    (NP←ADJ←NP T)
        --  (QUOTE (ABOUT X (# 2 1 TOPIC)))  )

(R:PP
    (NP←N T)
    (NP←PP T)
        --  (PROG (TEMPRELS) (SETQ TEMPRELS (QUOTE ((S REL (NP (DET
WHI) (N (# 1 1 HEAD)) (NU SG)) (VP (V BE) (PP (PREP (# 2 1 HEAD))
(# 2 2 IDENTITY)))))))) (RELTAG TEMPRELS) (RETURN (SEM (CAR TEMPRELS))))
  )
```

```
(R:REL
    AND
    (NP←REL (RELTAG (# 1)))
    (NOT (NP←N (MEM 1 DOCUMENT)))
        --  (PRED (# 1 1))  )

(REFRULE
    (T T)
        --  (QUANT (FOR EVERY X / DOCUMENT : (ABOUT X (# Ø TOPIC))
; (PRINTOUT X)))  )

(REFRULE?
    (T (INTERP P))
    (NOT (NP←NPR T))
        --  (QUOTE (# Ø))  )

(S:AND
    AND
    (S←AND T)
        --  (PRED (# 1 1))  )

(S:BE-INTERESTED
    (S←V (EQU 1 BE))
    (S←ADJ (EQU 1 INTERESTED))
    (S←PP (EQU 1 IN))
        --  (PRED (PRINTOUT (# 3 2 REFS?)))  )

(S:DCL
    (S←DCL-S T)
        --  (PRED (# 1 1))  )

(S:GIVE
    (S←V (MEM 1 GIVE))
    (S←OBJ T)
        --  (PRED (PRINTOUT (# 2 1 REFS?)))  )

(S:LIKE
    (S←V (MEM 1 LIKE))
    (S←COMPL←OBJ T)
        --  (PRED (PRINTOUT (# 2 1 REFS?)))  )

(S:IMP
    (S←IMP-S T)
        --  (PRED (DO (# 1 1)))  )

(S:NPU
    (S←NPU T)
        --  (PRED (PRINTOUT (# 1 1 REFS?)))  )
```

```
(S:OR
    OR
    (S+OR T)
        --  (PRED (# 1 1))  )

(S:PAPER-HAVE
    (S+NP (MEM 1 DOCUMENT))
    (S+V (EQU 1 HAVE))
    (S+OBJ T)
        --  (PRED (PRINTOUT (# 3 1 REFS?)))  )

(S:SEARCH
    (S+V (MEM 1 SEARCH))
    (S+PP (OR (EQU 1 FOR)
              (EQU 1 IN)
              (EQU 1 ON)))
        --  (PRED (PRINTOUT (# 2 2 REFS?)))  )

(S:WHQ
    (S+Q-S (LEAFMEMB P (QUOTE (WHQ WHICHQ WHEN WHERE WHY HOW HOWMANY))))
        --  (PRED (# 1 1))  )

(S:YES/NO
    (S+Q-S (NOT (LEAFMEMB P
                          (QUOTE (WHQ WHICHQ WHEN WHERE WHY HOW HOWMANY)
                                 ))))
        --  (PRED (TEST (# 1 1)))  )

(SS30
    (S+NP-V (EQU 2 BE))
    (S+OBJ (OR (EQU 1 WHQ THING SG)
               (EQU 1 WHQ THING PL)
               (EQU 1 WHQ THING SG/PL)
               (EQU 1 WHAT)))
        --  (PRED (PRINTOUT (1 . 1)))  )

(SS31
    (S+NP-V (AND (EQU 2 BE)
                 (NOT (OR (EQU 1 WHQ THING SG)
                          (EQU 1 WHQ THING PL)
                          (EQU 1 WHQ THING SG/PL)
                          (EQU 1 WHAT)))))
    (S+OBJ (NOT (OR (EQU 1 WHQ THING SG)
                    (EQU 1 WHQ THING PL)
                    (EQU 1 WHQ THING SG/PL)
                    (EQU 1 WHAT))))
        --  (PRED (EQUAL (1 . 1) (2 . 1)))  )
```

```
(SS32
    (S+NP-V (AND (MEM 1 INTEGER)
                 (EQU 2 BE)))
    (S+COMP (AND (OR (EQU . AT LEAST)
                     (EQU . AS MANY AS)
                     (EQU . ATLEAST)
                     (EQU . ASMANYAS))
                 (MEM 2 INTEGER)))
        --  (PRED (NOT (GREATER (2 . 2) (1 . 1))))  )

(SS33
    (S+NP-V (AND (MEM 1 INTEGER)
                 (EQU 2 BE)))
    (S+COMP (AND (OR (EQU . MORE THAN)
                     (EQU . MORETHAN))
                 (MEM 2 INTEGER)))
        --  (PRED (GREATER (1 . 1) (2 . 2)))  )

(SS34
    (S+NP-V (AND (MEM 1 INTEGER)
                 (EQU 2 BE)))
    (S+COMP (AND (EQU 1 EXACTLY)
                 (MEM 2 INTEGER)))
        --  (PRED (EQUAL (1 . 1) (2 . 2)))  )

(SS35
    (S+NP-V (AND (MEM 1 INTEGER)
                 (EQU 2 BE)))
    (S+COMP (AND (OR (EQU . FEWER THAN)
                     (EQU . LESS THAN)
                     (EQU . FEWERTHAN)
                     (EQU . LESSTHAN))
                 (MEM 2 INTEGER)))
        --  (PRED (GREATER (2 . 2) (1 . 1)))  )

(SS36
    (S+NP-V (AND (MEM 1 INTEGER)
                 (EQU 2 BE)))
    (S+COMP (AND (OR (EQU . AT MOST)
                     (EQU . ATMOST))
                 (MEM 2 INTEGER)))
        --  (PRED (NOT (GREATER (1 . 1) (2 . 2))))  )

(SS37
    (S+Q-S (NOT (LEAF P (QUOTE (WHQ WHICHQ WHEN WHERE WHY HOW HOWMANY)))
                ))
        --  (PRED (TEST (1 . 1)))  )

(SS38
    (S+DCL-S T)
        --  (PRED (1 . 1))  )
```

```
(SS39
    (S←NEG-S T)
        --  (PRED (NOT (1 . 1)))  )

(SS40
    (S←Q-S (LEAF P (QUOTE (WHQ WHICHQ WHEN WHERE WHY HOW HOWMANY))))
        --  (PRED (1 . 1))  )

(SS41
    (S←NP-V (OR (EQU 2 BE)
                (EQU 2 EXIST)))
        --  (PRED (EXIST (1 . 1)))  )

(TOPIC\ADJ
    AND
    (NP←ADJ (AND (NOT (MEM 1 DOCUMENT))
                 (NOT (EQ (CAR (# 1))
                          (QUOTE NP)))
                 (NOT (MEM 1 PADDING))))
        --  (QUOTE (# 1 1 TERM))  )

(TOPIC\ADJ-N
    OR
    (NP←ADJ-N (NOT (AND (OR (MEM 1 DOCUMENT)
                            (MEM 1 PADDING))
                        (MEM 2 DOCUMENT))))
    (OR (NP←PP T)
        (NP←REL T)
        (NP←COMPL T))
        --  (ADJPHRSE (QUOTE (# 0 IDENTITY)))  )

(TOPIC\ADJ-NPR
    OR
    (NP←NPR T)
    (NP←ADJ T)
    (OR (NP←PP T)
        (NP←REL T)
        (NP←COMPL T))
        --  (ADJPHRSE (QUOTE (# 2 IDENTITY)))  )

(TOPIC\ADJ←NP
    AND
    (NP←ADJ←NP T)
        --  (QUOTE (# 1 1 TOPIC))  )

(TOPIC\AND-NP
    AND
    (NP←AND T)
        --  (QUOTE (# 1 1 TOPIC))  )
```

```
(TOPIC\AND-S
    AND
    (S←AND T)
        --  (QUOTE (# 1 1 TOPIC))  )

(TOPIC\ESP
    (NP←ADVP (MEM 1 TRANSADV))
        --  (*FLAG (QUOTE (# 1 2 TOPIC)))  )

(TOPIC\N
    (NP←N (AND (NOT (MEM 1 PADDING))
               (NOT (MEM 1 DOCUMENT))))
        --  (QUOTE (# 1 1 TERM))  )

(TOPIC\NOM
    (NP←NOM T)
        --  (QUOTE (# 1 1 TOPIC))  )

(TOPIC\NR←S
    (NP←NR←S T)
        --  (QUOTE (# 1 1 TOPIC))  )

(TOPIC\NR←NP
    (NP←NR←NP T)
        --  (QUOTE (# 1 1 TOPIC))  )

(TOPIC\NOT-NP
    (NP←NEG T)
        --  (QUOTE (NOT (# 1 1 TOPIC)))  )

(TOPIC\NOT-S
    (S←NEG T)
        --  (QUOTE (NOT (# 1 1 TOPIC)))  )

(TOPIC\NPR
    (NP←NPR T)
        --  (QUOTE (# 1 1 TERM))  )

(TOPIC\OR-NP
    OR
    (NP←OR T)
        --  (QUOTE (# 1 1 TOPIC))  )

(TOPIC\OR-S
    OR
    (S←OR T)
        --  (QUOTE (# 1 1 TOPIC))  )
```

```
(TOPIC\PP
    AND
    (OR (NP-PP (NOT (EQ (CAADDR (# 2))
                        (QUOTE COMPL))))
        (NP-PP-AND-PP (NOT (EQ (CAADDR (# 2))
                               (QUOTE COMPL)))))
        --  (QUOTE (# 1 2 TOPIC))  )

(TOPIC\REL
    AND
    (NP-REL T)
        --  (QUOTE (# 1 1 TOPIC))  )

(TOPIC\S-COMPL
    (S-COMPL T)
        --  (QUOTE (# 1 1 TOPIC))  )

(TOPIC\S-NP
    (S-NP (AND (NOT (MEM 1 PADDING))
               (NOT (MEM 1 DOCUMENT))))
    (NOT (S-PRO T))
        --  (QUOTE (# 1 1 TOPIC))  )

(TOPIC\S-OBJ
    (OR (S-OBJ (AND (NOT (MEM 1 PADDING))
                    (NOT (MEM 1 DOCUMENT))))
        (S-OBJ-REL T))
        --  (QUOTE (# 1 1 TOPIC))  )

(TOPIC\S-PP
    AND
    (OR (S-PP T)
        (S-PP-AND-PP T))
        --  (QUOTE (# 1 2 TOPIC))  )

(TOPIC\ITEM
    (OR (NP-N (AND (LESSP (LENGTH (KEYPHRASE (LIST P)))
                          5)
                   (GREATERP (LENGTH (KEYPHRASE (LIST P)))
                             1)
                   (NOT (MEM 1 PADDING))
                   (NOT (MEM 1 DOCUMENT))))
        (NP-NPR (AND (LESSP (LENGTH (KEYPHRASE (LIST P)))
                            5)
                     (GREATERP (LENGTH (KEYPHRASE (LIST P)))
                               1))))
        --  (KEYPHRASE (LIST (QUOTE (# 2 IDENTITY))))  )
```

```
(TOPIC\TERM2
    (NP+N (AND (LESSP (LENGTH (KEYPHRASE (LIST P)))
                      5)
               (GREATERP (LENGTH (KEYPHRASE (LIST P)))
                         1)
               (OR (MEM 1 PADDING)
                   (MEM 1 DOCUMENT))))
    (NP+ADJ (AND (NOT (MEM 1 PADDING))
                 (NOT (MEM 1 DOCUMENT))))
        --  (KEYPHRASE (LIST (QUOTE (# 0 IDENTITY))))  )

(N:DOCUMENT
    (NP+N (MEM 1 DOCUMENT))
        --  (QUOTE DOCUMENT)  )

(S:CONCERN
    (S+NP (MEM 1 DOCUMENT))
    (S+V (MEM 1 CONCERN))
    (S+OBJ T)
        --  (PRED (PRINTOUT (# 3 1 REFS?)))  )

(S:PERTAIN
    (S+NP (MEM 1 DOCUMENT))
    (S+V (MEM 1 PERTAIN))
    (S+PP (OR (EQU 1 WITH)
              (EQU 1 TO)))
        --  (PRED (ABOUT (# 1 1) (# 3 2 TOPIC)))  )

(S:BE-ABOUT
    (S+NP (MEM 1 DOCUMENT))
    (S+V (EQU 1 BE))
    (S+PP (OR (EQU 1 ON)
              (EQU 1 ABOUT)
              (EQU 1 TO)
              (EQU 1 FOR)))
        --  (PRED (PRINTOUT (# 3 2 REFS?)))  )

(R:DOC
    (NP+N (MEM 1 DOCUMENT))
        --  (QUOTE (ABOUT X (# 0 TOPIC)))  )

(TOPIC\AUTHOR
    AND
    (OR (S+AND+NPR T)
        (S+NPR T))
    (S+V (MEM 1 WRITE))
        --  (AUTHOR: (QUOTE (# 1 1 TERM)))  )

(TOPIC\AUTHOR2
    AND
    (NP+N (MEM 1 DOCUMENT))
    (NP+PP+NPR (EQU 1 BY))
        --  (AUTHOR: (QUOTE (# 2 2 TERM)))  )
```

```
(TOPIC\NP+COMPL
    (NP+COMPL T)
        --  (QUOTE (# 1 1 TOPIC))  )

(TOPIC\PP+COMPL
    (NP+PP+COMPL T)
        --  (QUOTE (# 1 1 TOPIC))  )

(TOPIC\V-INTRANS
    (NOT (S+OBJ T))
    (S+V T)
        --  (APPEND (LIST (QUOTE V:)) (QUOTE (# 2 1 TERM)))  )

(TOPIC\V-TRANS
    OR
    (OR (S+OBJ+AND T)
        (S+OBJ+OR T))
    (S+V T)
        --  (APPEND (APPEND (LIST (QUOTE VP:)) (LIST (NFLCT-ING (CAR
(QUOTE (# 2 1 TERM))))))) (ADJPHRSE (QUOTE (# 1 1 IDENTITY))))  )

(TOPIC\V-TRANS2
    OR
    (OR (S+OBJ+AND T)
        (S+OBJ+OR T))
    (S+V T)
        --  (APPEND (APPEND (LIST (QUOTE VP:)) (LIST (NFLCT-ING (CAR
(QUOTE (# 2 1 TERM)))))))) (ADJPHRSE (QUOTE (# 1 1 IDENTITY))))  )

(S:RUN-A-SEARCH
    (S+V (MEM 1 RUN))
    (S+OBJ (MEM 1 SEARCH))
    (S+OBJ+PP T)
        --  (PRED (PRINTOUT (# 3 2 REFS?)))  )

(S:TOPIC
    (S+NP (NOT (MEM 1 DOCUMENT)))
    (S+V T)
        --  (QUANT (FOR EVERY X1 / DOCUMENT : (ABOUT X1 (# Ø TOPIC))
```

```
; (PRINTOUT X1)))  )

(TOPIC\EMPHASIS
    (NP*PP (MEM 2 EMPHASIS))
    (NP*PP*PP T)
        --  (*FLAG (QUOTE (# 2 2 TOPIC)))  )

(TOPIC\ADJ:SUPER
    AND
    (NP*DET*POSTART (MEMB (QUOTE SUPERLATIVE)
                          (CADR (# 1))))
    (NP*N T)
        --  (A[illegible] (LIST (NFLCT-ADJ (CADR (QUOTE (# 1 1 TERM))) (QUOTE
SUPERLATIVE))) (QU[illegible]E (# 2 1 TERM)))  )

(TOPIC\ADJ:COMP
    AND
    (NP*ADJ*COMP T)
    (NP*N T)
        --  (APPEND (LIST (NFLCT-ADJ (CAR (QUOTE (# 1 1 TERM))) (QUOTE
COMPARATIVE))) (QUOTE (# 2 1 TERM)))  )
)
(PRINT (QUOTE FUNCTIONS:) T)
(PRINT (QUOTE RULEFNS) T)
(READQ RULEFNS (*FLAG ADJPHRSE AUTHOR: INFLECT KEYPHRASE NFLCT-ING
POSTARTPHRS NFLCT-ADJ RULES))
(DEFINEQ

(*FLAG
  (LAMBDA (X)
    (COND
      ((MEMB (CAR X)
             (QUOTE (AND OR NOT)))
        (CONS (CAR X)
              (INTO (CDR X)
                    (FUNCTION *FLAG))))
      (T (APPEND X (LIST (QUOTE *)))))))

(ADJPHRSE
  (LAMBDA (X)
    (KEYPHRASE (CDR (FIRSTPART X (CDR (MEMB (ASSOC (QUOTE N)
                                                   X)
                                            X)))))))

(AUTHOR:
  (LAMBDA (X)
    (APPEND (LIST (QUOTE AUTHOR:))
            X)))
```

```
(INFLECT
  (LAMBDA (WORD X FORM)
    (PROG (FORMS INFL)
          (SETQ FORMS (GET WORD (QUOTE INFLEX)))
      L1  (SETQ INFL (GET (CAR FORMS)
                          X))
          (COND
            ((NULL INFL)
              (RETURN NIL))
            ((MEMB FORM (CADR INFL))
              (RETURN (CAR FORMS)))
            (T (SETQ FORMS (CDR FORMS))
               (GO L1))))))

(KEYPHRASE
  (LAMBDA (TREELIST)
    (COND
      ((NULL TREELIST)
        NIL)
      ((AND (ASSOC (QUOTE AUX)
                   TREELIST)
            (MEMB (CAR TREELIST)
                  (QUOTE (Q REL IMP DCL NOM))))
        (KEYPHRASE (CDR TREELIST)))
      ((ATOM (CAR TREELIST))
        (CONS (CAR TREELIST)
              (KEYPHRASE (CDR TREELIST))))
      ((AND (MEMB (CAAR TREELIST)
                  (QUOTE (DET)))
            (MEMB (CAR (CADDAR TREELIST))
                  (QUOTE (POSTART))))
        (APPEND (KEYPHRASE (T:SONS (POSTARTPHRS (CADDAR TREELIST))))
                (KEYPHRASE (CDR TREELIST))))
      ((MEMB (CAAR TREELIST)
             (QUOTE (NU DET INS AUX NR)))
        (KEYPHRASE (CDR TREELIST)))
      (T (APPEND (KEYPHRASE (T:SONS (CAR TREELIST)))
                 (KEYPHRASE (CDR TREELIST)))))))
```

```
(NFLCT-ING
  (LAMBDA (X)
    (PROG (Y1 Y2)
          (SETQ Y1 (MKSTRING X))
          (RETURN (COND
                    ((NOT (ATOM (GETP X (QUOTE V))))
                      (INFLECT X (QUOTE V)
                                 (QUOTE PRESPART)))
                    ((STRPOS (QUOTE "*")
                             (MKSTRING (GETP X (QUOTE V))))
                      (SETQ Y2 (SUBSTRING Y1 1))
                      (MKATOM (CONCAT Y1 (GLC Y2)
                                         (QUOTE "ING"))))
                    ((OR (NOT (STREQUAL (SUBSTRING Y1 -1)
                                        (QUOTE "E")))
                         (STREQUAL (SUBSTRING Y1 -2 -1)
                                   (QUOTE "EE")))
                      (MKATOM (CONCAT Y1 (QUOTE "ING"))))
                    (T (MKATOM (CONCAT (SUBSTRING Y1 1 -2)
                                       (QUOTE "ING")))))))))

(POSTARTPHRS
  (LAMBDA (POSTART)
    (COND
      ((OR (MEMB (QUOTE MUCH)
                 POSTART)
           (MEMB (QUOTE MANY)
                 POSTART))
       (SETQ POSTART (LIST (CAR POSTART)
                           (CADR POSTART)))))))

(NFLCT-ADJ
  (LAMBDA (WORD DEGREE)
    (PROG (WRDSTRING L TYPE END ENDL)
          (SETQ WRDSTRING (MKSTRING WORD))
          (SETQ L (SUBSTRING WRDSTRING 1))
          (SETQ TYPE (GETP WORD (QUOTE ADJ)))
          (RETURN
            (COND
              ((OR (EQ TYPE (QUOTE *))
                   (EQ TYPE (QUOTE REG)))
                (COND
                  ((EQ DEGREE (QUOTE COMPARATIVE))
                    (MKATOM (CONCAT (QUOTE "MORE-")
                                    WRDSTRING)))
                  (T (MKATOM (CONCAT (QUOTE "MOST-")
                                     WRDSTRING)))))
              ((EQ TYPE (QUOTE IRR))
                (INFLECT WORD (QUOTE ADJ)
                         DEGREE))
```

```
;  <WOOD          ;  <WOODS>DDCRULES.;4   THU 1-JUL-71 5:40PM


                                (T (COND
                                     ((EQ DEGREE (QUOTE COMPARATIVE))
                                       (SETQ END (SUBSTRING TYPE 1
                                                            (SUB1 (STRPOS (QUOT
                                                                           TYPE)
                                     (T (SETQ END (SUBSTRING TYPE
                                                             (ADD1 (STRPOS (QUO
                                                                            TYPE
                                                             -1))))
                                   (SETQ ENDL (GLC L))
                                   (COND
                                     ((EQ ENDL (QUOTE Y))
                                       (MKATOM (CONCAT L (QUOTE "I")
                                                       END)))
                                     ((EQ (GNC TYPE)
                                          (QUOTE *))
                                       (MKATOM (CONCAT WRDSTRING ENDL
                                                       (SUBSTRING END 2 -1))))
                                     (T (MKATOM (CONCAT WRDSTRING END)))))))))
                (
                        (RULES
                          (LAMBDA (P TYPEFLAG)                          (* DETERMINES
                                                                        SEMANTIC RUL
                                                                        AND ALSO SE
                            (PROG (TYPE HEAD VAL)
                                  (COND
                                    ((NULL TYPEFLAG)
                                      (SETQ TYPEFLAG (CAR P))))
)                                 (COND
(PRINT (                            ((EQ TYPEFLAG (QUOTE NP))
(PRINT (                              (COND
(RPAQQ T  ;E                            ((ASSOC (QUOTE NPR)
NP←ADJ-A                                        (CONSTITUENTS P))
NP←AND N                                  (RETURN NPRRULE))
NP←N NP←  ;G                            (T (SETQ MODE (QUOTE FAIL)))))
NP←PP NP                            ((EQ TYPEFLAG (QUOTE VP))
NP←REL S                              (SETQ MODE (QUOTE AND)))
S←IMP-S   ·N                        ((EQ TYPEFLAG (QUOTE S))
S←OBJ←OR                              (SETQ MODE (QUOTE FAIL)))
S←Q S←Q-                            ((MEMB TYPEFLAG (QUOTE (N V NPR ADJ ADV)))
(DEFINEV                              (RETURN TERMRULE)))
(NP←ADVP                          (COND
                                    ((EQ (CAR P)
                                         (QUOTE NP))
(NP←PP←P  /4                          (SETQ QVAR (COND
                                                   ((GETTAG P (QUOTE QVAR ))
(NP←ADJ                                             (T (TAG P (QUOTE QVAR)
(NP←ADJ#                                                      (NXTVAR)))))
(NP←ADJ-  )J                        (PUT QVAR (QUOTE NODE)
                                         P)))
(NP←ADJ-
```

```
(NP←ADJ←COMP (NP ((ADJ ((COMPARATIVE))
                        1))))
(NP←ADJ-N (NP ((ADJ NIL 1)
               (N NIL 2))))
(NP←ADJ←N/N (NP ((ADJ ((N ((N NIL 1)
                           /
                           (N NIL 2)))))))))
(NP←ADJ←NP (NP ((ADJ ((NP NIL 1))))))
(NP←ADJ←NPR (NP ((ADJ ((NPR NIL 1))))))
(NP←AND (NP (AND (NP NIL 1))))
(NP←AND2 (NP (AND (NP NIL 1)
                  (NP NIL 2))))
(NP←COMPL (NP ((COMPL ((S NIL 1))))))
(NP←DET (NP ((DET NIL 1)
             (NU NIL 2))))
(NP←DET←COMP (NP ((DET ((COMP ((ADV NIL 1)
                               (NP ((NPR ((INTEGER NIL 2)))))))
                        MANY))
                  (NU NIL 3))))
(NP←DET←MANY (NP ((DET ((NP ((NPR ((INTEGER NIL 1)))))
                        MANY))
                  (NU NIL 2))))
(NP←DET←POSTART (NP ((DET ((POSTART NIL 1))))))
(NP←N (NP ((N NIL 1))))
(NP←NEG (NP (NEG (NP NIL 1))))
(NP←NOM (NP (NOM (S NIL 1))))
(NP←NPR (NP ((NPR NIL 1))))
(NP←NR←S (NP ((NR ((S NIL 1)))))))
(NP←NR←NP (NP ((NR ((NP NIL 1))))))
(NP←OR (NP (OR (NP NIL 1))))
(NP←OR2 (NP (OR (NP NIL 1)
                (NP NIL 2))))
(NP←PP←NPR (NP ((PP ((PREP NIL 1)
                     (NP ((NPR NIL 2)))))))))
(NP←PP (NP ((PP ((PREP NIL 1)
                 (NP NIL 2))))))
(NP←PP←COMPL (NP ((PP ((NP ((COMPL ((S NIL 1)))))))))))
(NP←PP←AND (NP ((PP ((PREP NIL 1)
                     (NP (AND (NP NIL 2)
                              (NP NIL 3)))))))))
(NP←PP←AND←PP (NP ((PP (AND (PP ((PREP NIL 1)
                                 (NP NIL 2)))))))))
(NP←PP←PP (NP ((PP ((NP ((PP ((PREP NIL 1)
                              (NP NIL 2)))))))))))
(NP←PP←PP←PP (NP
              ((PP ((NP ((PP ((NP ((PP ((PREP NIL 1)
                                        (NP NIL 2)))))))))))))))
(NP←PRO (NP ((PRO NIL 1))))
(NP←REL (NP ((S NIL 1))))
(S←ADJ (S ((VP ((ADJ NIL 1))))))
```

```
(S←AND (S (AND (S NIL 1))))
(S←COMP (S ((VP ((COMP ((ADV NIL 1)
                        (NP NIL 2)))))))
(S←COMPL (S ((VP ((COMPL ((NP (NOM (S NIL 1)))))))))
(S←COMPL←OBJ (S
                ((VP
                   ((COMPL ((NP (NOM (S ((VP ((NP NIL 1)))))))))))))
(S←DCL (S (DCL)))
(S←DCL-S (S (DCL (S NIL 1))))
(S←IMP (S (IMP)))
(S←IMP-S (S (IMP (S NIL 1))))
(S←NEG (S (NEG)))
(S←NEG-S (S (NEG (S NIL 1))))
(S←NP (S ((NP NIL 1))))
(S←NPR (S ((NP ((NPR NIL 1))))))
(S←AND←NPR (S ((NP (AND (NP ((NPR NIL 1)))))))
(S←NP-V (S ((NP NIL 1)
            (VP ((V NIL 2))))))
(S←NPU (S (NPU (NP NIL 1))))
(S←OBJ (S ((VP ((NP NIL 1))))))
(S←OBJ←AND (S ((VP ((NP (AND (NP NIL 1)))))))
(S←OBJ←OR (S ((VP ((NP (OR (NP NIL 1)))))))
(S←OBJ←PP (S ((VP ((NP ((PP ((PREP NIL 1)
                             (NP NIL 2)))))))))
(S←OBJ←REL (S ((VP ((NP ((S NIL 1)))))))
(S←OR (S (OR (S NIL 1))))
(S←PP (S ((VP ((PP ((PREP NIL 1)
                    (NP NIL 2)))))))
(S←PRO (S ((NP ((PRO NIL 1))))))
(S←PP←AND (S ((VP ((PP ((PREP NIL 1)
                        (NP (AND (NP NIL 2)
                                 (NP NIL 3)))))))))
(S←PP←AND←PP (S ((VP ((PP (AND (PP ((PREP NIL 1)
                                    (NP NIL 2)))))))))
(S←PP←PP (S ((VP ((PP ((NP ((PP ((PREP NIL 1)
                                 (NP NIL 2)))))))))))
(S←Q (S (Q (NP NIL)
           (VP NIL))))
(S←Q-NEG (S (Q NEG (NP NIL)
                (VP NIL))))
(S←Q-S (S (Q (S NIL 1))))
(S←V (S ((VP ((V NIL 1))))))
(S←VP (S ((VP NIL))))
)
(PRINT (QUOTE VALUES:) T)
(PRINT (QUOTE RULELISTS) T)
(RPAQQ RULELISTS (REFRULES PRERULES DRULES SETRUL NPRRULE TERMRULE
HEADRULES SPECIALTOPICRULE))
(DEFINEQ
```

```
(REFRULES (TOPIC\ADJ TOPIC\ADJ+NP TOPIC\AND-NP TOPIC\AND-S TOPIC\N
                     TOPIC\NOM TOPIC\NPR TOPIC\OR-NP TOPIC\OR-S
                     TOPIC\PP TOPIC\S+NP TOPIC\S+COMPL TOPIC\S+OBJ
                     TOPIC\S+PP TOPIC\S.V TOPIC\TERM TOPIC\NOT-S
                     TOPIC\NOT-NP TOPIC\AUTHOR TOPIC\AUTHOR2
                     TOPIC\EMPHASIS TOPIC\NP+COMPL TOPIC\PP+COMPL
                     TOPIC\V-INTRANS TOPIC\V-TRANS TOPIC\V-TRANS2
                     TOPIC\ADJ:COMP TOPIC\ADJ:SUPER S:TOPIC))
(PRERULES (S:AND S:OR S:DCL S:IMP S:WHQ S:YES/NO NIL PR1 NIL PR2 NIL
                 PR3 NIL PR4 NIL PR5 NIL S:NPU NIL PR6))
(DRULES (DD0 DD1 DD1:5 DD2 DD2:5 DD3 DD4 DD5 DD6 DD7 DD8 DD9 DD10 DD11
         DD12 DD13 DD14 DD15 D:ANAPHORA D:ALL\ONES))
(SETRUL (DD16))
(NPRRULE (NP:NPR))
(TERMRULE (ANY:TERM))
(HEADRULES (((S (NPU))
              (QUOTE NPU))
             ((S (AND))
              (QUOTE AND))
             ((S (OR))
              (QUOTE OR))
             ((S (Q (S NIL)))
              (QUOTE Q))
             ((S (Q (VP NIL 1)))
              (HEAD 1))
             ((S (DCL (S NIL)))
              (QUOTE DCL))
             ((S (DCL (VP NIL 1)))
              (HEAD 1))
             ((S (IMP (S NIL)))
              (QUOTE IMP))
             ((S (NEG (S NIL)))
              (QUOTE NEG))
             ((S (NEG (VP NIL 1)))
              (HEAD 1))
             ((S ((VP NIL 1)))
              (HEAD 1))
             ((VP ((VP NIL 1)))
              (HEAD 1))
             ((VP ((V NIL 1)))
              (TERM 1))
             ((NP ((NPR NIL 1)))
              (TERM 1))
             ((NPR NIL 1)
              (TERM 1))
             ((NP ((N NIL 1)))
              (TERM 1))
             ((N NIL 1)
              (TERM 1))
             ((NP (OR))
              (QUOTE OR))
```

```
            ((NP (AND))
             (QUOTE AND))
            ((ADJ NIL 1)
             (TERM 1))
            ((PREP NIL 1)
             (TERM 1))
            ((NP (NOM (S NIL 1)))
             (HEAD 1))
            ((V NIL 1)
             (TERM 1))
            ((NP ((PRO NIL 1)))
             (TERM 1))
            ((PRO NIL 1)
             (TERM 1))
            ((NP (NEG))
             (QUOTE NEG))
            ((ORD NIL 1)
             (HEAD 1))
            NIL))
(SPECIALTOPICRULE (S:TOPIC))
)
(PRINT (QUOTE VALUES:) T)
(PRINT (QUOTE (TOPICRULES)) T)
(DEFINEV
(TOPICRULES (TOPIC\NOT-NP TOPIC\NOT-S TOPIC\AUTHOR NIL
                          (OR TOPIC\TERM TOPIC\TERM2 TOPIC\ESP
                              TOPIC\EMPHASIS TOPIC\NR+S TOPIC\NR+NP
                              (AND TOPIC\OR-S TOPIC\AND-S TOPIC\OR-NP
                                   TOPIC\AND-NP TOPIC\ADJ+NP
                                   TOPIC\NP+COMPL TOPIC\S+COMPL
                                   TOPIC\PP TOPIC\AUTHOR2
                                   TOPIC\PP+COMPL TOPIC\REL
                                   (OR TOPIC\ADJ:COMP
                                       (AND TOPIC\ADJ (OR
                                                       TOPIC\ADJ:SUPER
                                                           TOPIC\N)
                                                   TOPIC\NPR)
                                            TOPIC\ADJ-N TOPIC\ADJ-NPR)
                                   TOPIC\NOM TOPIC\NR+S TOPIC\NR+NP
                                   TOPIC\S+NP TOPIC\V-INTRANS
                                   TOPIC\V-TRANS TOPIC\V-TRANS2
                                   TOPIC\S+OBJ TOPIC\S+PP)))
)
(PRINT (QUOTE VALUES:) T)
(PRINT (QUOTE MORPHOLOGY) T)
(RPAQQ MORPHOLOGY (MORPHTABLE MORPHTESTS))
(DEFINEV
```

```
(MORPHTABLE ((N ((S)
                 NIL N (PLURAL -S)
                 (NUMBER PL))
                ((E S)
                 NIL N (PLURAL -ES)
                 (NUMBER PL))
                ((I E S)
                 (Y)
                 N
                 (PLURAL -ES)
                 (NUMBER PL))
                ((E N)
                 NIL N (PLURAL -EN)
                 (NUMBER PL)))
            (V ((E S)
                NIL V (CTYPE ES-ED)
                (TNS PRESENT)
                (PNCODE 3SG))
               ((I E S)
                (Y)
                V
                (CTYPE ES-ED)
                (TNS PRESENT)
                (PNCODE 3SG))
               ((S)
                NIL V (OR (CTYPE S-ED)
                          (CTYPE S-D))
                (TNS PRESENT)
                (PNCODE 3SG))
               ((E D)
                NIL V (OR (CTYPE S-ED)
                          (CTYPE ES-ED))
                (TNS PAST)
                (PASTPART))
               ((I E D)
                (Y)
                V
                (CTYPE ES-ED)
                (TNS PAST)
                (PASTPART))
               ((E D)
                (E)
                V
                (CTYPE S-D)
                (TNS PAST)
                (PASTPART))
               ((I N G)
                NIL V (OR (CTYPE S-ED)
                          (CTYPE ES-ED))
                (PRESPART))
```

```
                ((I N G)
                 (E)
                 V
                 (CTYPE S-D)
                 (PRESPART)))
            (ADJ ((E R)
                  NIL ADJ (COMPFORM ER-EST)
                  (COMPARATIVE))
                 ((E R)
                  (E)
                  ADJ
                  (COMPFORM R-ST)
                  (COMPARATIVE))
                 ((I E R)
                  (Y)
                  ADJ
                  (COMPFORM ER-EST)
                  (COMPARATIVE))
                 ((E S T)
                  NIL ADJ (SUPFORM ER-EST)
                  (SUPERLATIVE))
                 ((E S T)
                  (E)
                  ADJ
                  (SUPFORM R-ST)
                  (SUPERLATIVE))
                 ((I E S T)
                  (Y)
                  ADJ
                  (SUPFORM ER-EST)
                  (SUPERLATIVE)))))
(MORPHTESTS ((INTEGER ((NUMBERP *)
                       (LIST *)))
             (NPR ((OR (NUMBERP *)
                       (TIMEP *)
                       (AND (LISTP *)
                            (MEMB (CAR *)
                                  (QUOTE (* **)))))
                       (CONTRACTP *))
                  (LIST *)))
             (LIST ((AND (LISTP *)
                         (NOT (MEMB (CAR *)
                                    (QUOTE (* **)))))
                    (LIST *)))
             (ADJ ((HYPHENADJ *)
                   (LIST *)))))
)
STOP
```

APPENDIX D: DOCUMENTATION OF FUNCTIONS

This appendix is designed to give a detailed description of the complete set of functions involved in the parser, interpreter, and grammar of the system. It describes each function, places it the overall framework of the system, explains how it interacts with other functions, and describes the functions of various arguments and temporary storage locations. Together with the listings of the programs, grammar, and semantic rules, it provides a thorough documentation of the English preprocessor system.

Each function is given with the names of its arguments in the form of a typical call to the function (e.g. (#N) is a call to the function # with argument N). Those functions which take an indefinite number of arguments bound as a list to a single atom are listed as a dotted pair of the function and the argument list (e.g. (BUILD . ARGS) represents a call to the function BUILD with the CDR of the calling form bound to ARGS).

ENGLISH PROCESSING FUNCTIONS

(# N)

The function # is used in templates of semantic rules to reference the node of the tree that matches the node numbered N in the tree fragment used for the match. For example, in a template (NP:N (TESTFN (# 1))), the expression (# 1) will evaluate to the node of the tree which matches node 1 of the fragment NP:N. TESTFN in this case is a hypothetical condition which is to be true of noe (# 1).

(ABORT)

ABORT is a function for terminating fruitless paths in a parsing. It is used as an action on arcs of the grammar (usually embedded in a COND) to abort the arc if some condition is not satisfied.

(ACT ACTIONS NONTERMFLAG)

ACT is the function called by STEP to execute the actions on the arcs of the grammar. ACTIONS is the list of actions to be performed, and normally it is terminated by a terminating action (JUMP, TO, or ABORT). However, there are cases (e.g. on a JUMP arc) when ACT is called with a list of actions which is not so terminated. In this case, the argument NONTERMFLAG is set to permit a normal return from ACT. If NONTERMFLAG is not set and a non-terminated list of actions is encountered, then there is a bug in the grammar and ACT prints an error message.

Terminating actions are indicated by returning a value

*L0, *L1, *L2, *END, *HELP, indicating a location in the STEP routine to which control is to resume. Terminal acts other than ABORT also have the responsibility of setting up the configuration on which STEP is to operate.

(ADDL REG EXPRESSION)

ADDL is a function for use on arcs of the grammar for adding to the contents of a register. It adds the results of evaluating EXPRESSION to the left of the contents of the register REG (which must be a list).

(ADDR REG EXPRESSION)

ADDR is like ADDL except that it adds information to the right end of the list in register REG.

(ALT:STACK ALT)

This function extracts the stack from an alternative ALT. It is used in several places by other functions. It is one of a large class of such functions which have been named to aid the readibility of the programs. All functions containing a colon in their names in this fashion are functions for extracting information from a list that is functioning as a special "data type". The portion of the name before the colon names the data type to which the function applies, and the portion of the name which follows the colon names the "field" of the data type whose value is being extracted. Thus an expression of the form (ALT:STACK X) embedded in a piece of code tells the reader both that X is an ALT (the data type for alternatives in the grammar) and that the value of

the expression will be the stack which was saved in that alternative.

(ALT:STATE ALT)

This function is similar to ALT:STACK, but extracts the state of the alternative.

(ALT:STRING ALT)

This function extracts the string from an alternative.

(ALT:WEIGHT ALT)

Extracts the weight associated with an alternative. (The weight of an alternative is a measure of how "likely" the alternative is.)

(ALTARCGEN)

This function is called in a number of places in the STEP function to store ALTARC alternatives. If there are arcs which are as yet untried (and if LEXMODE is not set) then an alternative is stored which will enable those arcs to be tried later if the current path is not successful. (LEXMODE is set during the processing of certain parts of reduced conjunctions, when the parser is constrained to follow the trail left by a previous parsing of the same string, and ALTARC alternatives are not generated.)

(ALTCONJGEN PPATH)

ALTCONJGEN generates ALTCONJ alternatives for restarting a previous configuration of the parse on the string which follows a

conjunction (it is called by SYSCONJ as part of the system facility for handling reduced conjunctions). PPATH is the partial path entry from some previous configurations which saves the necessary information for restarting.

(ASSIST)

ASSIST is a function called by parser when no parsings have been found and no more alternatives remain to be tried. When the flag ASSIST FLAG is not set, it has no effect. Otherwise, it locates the blocked configuration which got the farthest into the input string before blocking, and executes a call to HELPER to allow the user to investigate. (It identifies itself with the typed message "ASSISTANCE:"). This function is used only for system debugging, and ASSISTFLAG would normally be off for users. If the call to HELPER is terminated with RPT instead of OK, ASSIST will attempt to resume the parsing from that point.

(BACKUP)

This function is designed for use in an ASSIST break (i.e. a call to HELPER from ASSIST). It allows the user to back up the configuration along the path leading to it. Thus the user can back up along the computation of the blocked configuration (and by terminating the call to HELPER with RPT he can restart the computation from the configuration to which he has backed up). Each call to BACKUP backs up the configuration one step and prints out the state of the configuration after the backup.

(BOOLGET X)

This function is used in the generation of Boolean

expressions from semantic interpretations. It searches the Boolean expression X for instances of the predicate ABOUT (which represents the predication of a document being about a topic) and gathers up the corresponding Boolean expression of topics. This allows for the eventual possibility of inclusion in the semantic interpretations of predications involving predicates other than ABOUT for dealing with non content indicative items in requests.

(BOOLREQ CNF)

BOOLREQ converts a normal conjunctive normal form Boolean expression CNF into a form which approximates the DDC retrieval language, by sorting negations to the end and raising negations if necessary so that they are always components of an AND and not of an OR. The latter is done in order to provide a Boolean request which can always be done by intersecting inverted files and never requires constructing the complement of an inverted file. BOOLREQ1 performs the bulk of this operation. NEGSORT sorts the NOT's to the ends of clauses, and converts such clauses to instances of the operator SDIFF (which represents the operation of taking the set difference between two Boolean expressions).

(BOOLREQ1 CLAUSE)

See BOOLREQ

(BOOLRET BOOLEXP)

In an eventual complete system in which the English preprocessor would be interfaced to the actual DDC retrieval system,

this would be the interfacing function which would pass the Boolean expression BOOLEXP to the retrieval system and initiate retrieval. In the current system it merely types out the Boolean expression on the teletype.

(BUILD . ARGS)

BUILD is a function which can be used on the arcs of the grammar to build sentence structure. It takes an indefinite number of arguments, the first of which is the name of a structure fragment. The remaining arguments of BUILD (if any) are items to be substituted for specially marked leaves in the structure fragment. The structure fragment is processed from left to right, and when a node + is encountered the next item in the remaining argument list is taken as the name of a register whose value is to be inserted for the symbol +. When a node # is encountered, the next item is taken as a form to be evaluated, and the resulting value is substituted for the symbol #. In addition, when the symbol * is encountered, the current value being scanned by the pointer * is copied into the structure, and where subexpressions of the form (@ X1 X2 .Xn) are encountered, a single list is generated which is the result of appending X1, X2, ... Xn (which must be lists) into a single list. BUILD1 and BUILD2 are the functions that do most of the work.

(BUILD1 X)

See BUILD.

(BUILD2 X)

See BUILD.

(BUILDQ . ARGS)

BUILDQ is like BUILD except that its first argument is taken literally as the structure fragment $ while BUILD's first argument is evaluated.

(CAT CATEGORY)

CAT is a function for use in conditions on arcs in the grammar for testing the syntactic categories of X the current word being scanned. CATEGORY can be either a single syntactic category name or a list of category names. CAT is true if the current word can be in the indicated category or one of the indicated categories.

(CATCHECK CATEGORY FLAG)

CATCHECK is the function used by CAT and by STEP for accessing the dictionary to determine whether the current word can be a member of a particular category. It returns a list (called a form-features list) whose first member is the standard (uninflected or "root") form of the word as used in this category and whose remaining elements are inflectional features associated with this word used as this category. In addition, if there are other such lists (corresponding to different senses of the word) for the same syntactic category and the flag FLAG is set, then an ALTCAT alternative is generated to enable that alternative to be pursued later. This flag is set when STEP calls CATCHECK for a CAT arc, but is not set for other uses.

(CCHECK TEMPLATE MLIST)

CCHECK is the function which checks semantic conditions

in templates during the matching of semantic rules by the interpreter. TEMPLATE is the template in question, and MLIST is a LIST of possible matches for the template which are to be screened by CCHECK. Each element of MLIST is an ALIST whose elements are dotted pairs of node numbers in the template and their corresponding matches in the tree.

(CHANGEWORD . ARGS)

CHANGEWORD is a function for use in a REQUESTDEF break when the system encounters an unknown word. It takes an indefinite number of arguments ARGS, and it patches this list into the input string in place of the current word. If ARGS is NIL, then the current word is deleted. It performs the necessary side effects to enable the parsing to continue as if the resulting string were the one originally typed.

(CHECKF CATEGORY FEAT)

CHECKF is a function which checks the current word (*) to see if it has feature FEAT under syntactic category CATEGORY in the dictionary. FEAT may be a list of features instead of a single feature, in which case CHECKF is true if * has any of the indicated features. The dictionary checking is actually performed by CHECKF1.

(CHECKF1 CATEGORY FEAT)

See CHECKF.

(CNF BOOLEXP)

CNF is the function which converts Boolean expressions to

conjunctive normal form. The result of CNF is a list of lists of keyphrases or their negations. Each element of the top level list is a disjunction (OR) of its contained phrases, while the top level list is a conjuntion (AND) of these disjunctions, although the AND's and OR's are implicit in the list structure and do not explicitly appear. BOOLREQ uses the output of CNF to construct its Boolean request.

(COMPFORM ENDING)

COMPFORM is a function to check whether the given ending is the indicated ending form the comparative form of the adjective *. It is used in MORPHTABLE for the morphological analysis of comparative adjectives.

(COMPLEMENT LITERAL)

This function returns the complement of the literal keyphrase LITERAL. If LITERAL is negated, then COMPLEMENT drops the negation; otherwise, it adds a negation. The function is used in BOOLREQ1.

(CONJOIN)

This function is one of three primary functions in the SYSCONJ facility. (The functions are CONJOIN, POPCONJ, and SYSCONJ.) It is never called explicitly from any part of the code, but a call to CONJOIN is placed on the stack by SYSCONJ along with the current status of the configuration which is being suspended in order to restart an earlier configuration on the string which follows the conjunction. When the restarted configuration has completed the construction which it was building, control will pop to this special stack entry, and the function CONJOIN will be executed to resume the suspended configuration on some tail of the string consumed by

the restarted configuration. CONJOIN enumerates all possible such tails, and generates alternatives for each of them. It also gathers up multiple conjuncts into a single level conjunction (i.e. (AND A B C) instead of (AND A (AND B C))--this is done by the first COND in the program) and it uses a heuristic strategy for selecting the "preferred" tail to try first. The strategy is that if the word which immediately preceded the conjunction word (i.e. (CAR (PATH:STRING (CAAR TEMP)))) is repeated somewhere in the string consumed by the restarted configuration, then the preferred place to resume the suspended alternative is immediately after this word. The location of such an alternative is performed by the loop at L1.

(CONJSCOPE SCOPEWORD CONJ)

This function is a predicate which is true of a conjunction and its left-hand scope indicator. That is, CONJSCOPE is true when SCOPEWORD is "both" and CONJ is "and" or when SCOPEWORD is "either" and CONJ is "or". It is used in CONJSTARTS for selecting preferred restart configurations.

(CONJSTARTS PPATH STATES)

This function computes restart configurations for SYSCONJ. It returns a list of the possible restart configurations ordered with the most likely one last, so that when they are placed on the ALTS list in the order given, the most likely one will be the most recent ALT. CONJSTARTS operates by backing up the partial path (PPATH) leading to the configuration where the conjunction was encountered and picking up possible restart configurations. It is forced to back up across at least one word in the string, and the flag

FIRSTFLAG is used to remember whether this condition has been met. It backs up along the partial path at each level until it is exhausted and then goes up the stack one level and starts backing across that level. It is forbidden, however, from backing up the stack beyond a previous SYSCONJ entry. When such a condition is encountered, or if the stack is emptied, then QUITFLAG is set to indicate that all possible configurations have been found. It will not generate a restart configuration which was reached as a result of a JUMP arc, since that would duplicate a configuration which can be reached by restarting the configuration at the beginning of that JUMP arc. Also, if CONJSTARTS is given a list of states as its second argument STATES, then only restart configurations in those states will be considered; otherwise, any state is possible.

CONJSTARTS uses several heuristics to locate preferred restart configurations. In the course of operation it selects up to three preferred alternatives (PREFERRED$\emptyset$, PREFERRED1 and PREFERRED2, favored in that order). PREFERRED$\emptyset$ is that alternative, if any, which is indicated by a scopeword for the current conjunction ("both" for "and" or "either" for "or"). PREFERRED1 is the alternative that is indicated by a repeated word-- i.e. when the word which immediately follows the conjunction occurs in the preceding string. PREFERRED2 is the alternative which is the beginning of the current constituent being built.

(CONSTITUENTS NODE)

CONSTITUENTS s the function which when applied to a node of a parse tree yields a list of the immediate constituents (daughters) of that node. For the tree notation currently in use,

(CONTRACTP WORD)

CONTRACTP is a function for morphological analysis of words which appear to be contract numbers. It is only a crude approximation to an actual recognizer of contract numbers. It is only called for words which are not already in the dictionary, and hence will not cause any conflicts with words which are entered in the dictionary that might meet its conditions but not be contract numbers. The conditions are that the word contains at least one hyphen, one numerical digit, one alphabetic letter, and no other punctuation marks.

(CTYPE ENDING)

CTYPE is a predicate used in MORPHTABLE entries for determining whether the conjugation type of the current word* is the same as that given as the argument ENDING.

(DDEF . ARGS)

DDEF is a function for adding entries to the dictionary. It is not called explicitly by any functions, but is intended for use by a user or systems programmer at the executive level or in a break. ARGS should be a list whose first element is the word to be defined, and whose remaining elements in pairs are property names and property values to be added to the dictionary entry for the word. It also adds the word to the global list DICTIONARY.

(DEFAULTSEM P TYPEFLAG)

This function is used in the semantic interpretation system to provide the default interpretation T for the restrictions on

the range of quantification of a noun phrase, when there are no restrictions implied by the RRULES. It is called by INTERP.

(DETBUILD)

DETBUILD is a function which is called by arcs of the grammar to build the determiner structure of a noun phrase. It combines the contents of the registers POSTART and DET with the appropriate structure.

(DETOUR)

DETOUR is the function which chooses the next alternative to be tried when the parser encounters a dead end or is instructed to find another parsing. It searches for the most recent alternative with the lowest weight.

(DICT? WORD)

DICT? is a function for examining the dictionary entry for a word WORD. It is not called by any function, but is intended for use by a user or systems programmer

(DICTCHECK LEX CATEGORY)

This function checks the dictionary entry for the word LEX to see if it can be analyzed as a member of the syntactic category CATEGORY. It returns a list is a list consisting of a standard (root) form of the word and the inflectional features associated with that word as an inflected form of the indicated root. This is the function which decodes the various types of abbreviated

dictionary formats into this standard list of form-features lists.

(EQU . ARGS)

EQU is a function for use in the templates of semantic rules. ARGS is a list whose first element is a number. EQU checks whether the terminal string dominated by the node corresponding to this number in the template match is identical with the remainder (CDR) of the list ARGS.

(EVALLOC FORM)

EVALLOC is a function which is used by several of the functions which are used by the grammar. FORM is an argument list for some function, and EVALLOC decides whether the argument is the name of a register (in which case it applies GETR to get the contents of the register) or a form for EVAL in which case it calls EVAL. It also handles the evaluation of the special current constituent pointer *, and fills the current constituent in as a default for certain cases where argument is missing.

(EXECUTE FORM)

This function is used for execution of query language expressions. In the BBN implementation it provides for the writing of the answer to a file and giving the user an option of taking it offline or online after giving him a measure of the size of the answer. In the 1108 system it merely provides a local variable COUNT to make the retrieval functions which refer to it work properly, and calls EVAL.

(EXIST X)

EXIST is a predicate which is universally true of any argument. It is used for the interpretation of verbs which are synonymous with the English word "exist".

(FOR QUANT *X* / CLASS : PX ; QX)

FOR is the function which in a full blown system would perform quantification in the retrieval component. In the current DDC system, the only types of requests which are meaningful are requests which quantify over documents which are about some topic. Thus, the function verifies that CLASS is the class DOCUMENT (it gives an appropriate diagnostic otherwise), and it scans the restrictions on the range of quantification (PX) and the quantified form (QX) for instances of the predicate ABOUT (which asserts that a document is about a topic). If there are no instances, then the system prints an error diagnostic and quits; otherwise, it uses BOOLGET to obtain the Boolean combination of all such instances of ABOUT, CNF to convert this to conjunctive normal form, BOOLREQ to convert that to an approximation of the DDC query language, and BOOLRET to print it out.

(GETF FEATURE)

GETF is a function which obtains the value of a feature FEATURE for the current word on a CAT arc. As a side effect of the call to CATCHECK on a CAT arc, the atom FEATURES is bound to the list of inflectional features associated with the current inflected word. GETF accesses features from this list.

(GETR REG WHERE)

GETR is a function for getting the contents of a register REG. During the parsing, the contents of the registers are kept on a list REGS of alternating register names and register values. GETR searches this list for the register REG. However, there are times when one wants to get the contents of a register at some higher level on the stack. The argument WHERE allows for the specification of the stack location. If WHERE is T, then the top level is used. If it is "NEAREST" then GETR searches the current level and then successively looks up the stack until it finds an instance of REG. If WHERE is a number, then GETR looks at that numerical stack position. Otherwise WHERE can be a condition on the STATE, REGS, and ACTIONS of a stack entry which determines the level of the stack to use.

(GETREFS P REFLISTS)

GETREFS is a function used in the semantic interpretation component by the function SORTREFS, which sorts the list of sub nodes to be interpreted into left-to-right order. It serves double duty as a predicate indicating whether P is a node which is to be interpreted, and if so it returns the list of the alternative reflists which belong to that node P. For more detail, see SORTREFS.

(GETROOT WORD CATEGORY)

GETROOT is a function for obtaining the root form of the word WORD viewed as a member of category CATEGORY. Note: if there are several possible roots, only the first one in the dictionary is found. e.g. (GETROOT (QUOTE SAW) V) will return SAW or SEE depending

on which is first in the dictionary entry under the category V.

(GETTAG P TAGNAME)

GETTAG is a function for retrieving items from TAGLIST, a global variable which holds tags associated with nodes in the tree and behaves like a property list for tree nodes. GETTAG returns the value of the tag TAGNAME for the node P.

(HEAD P)

HEAD is the function which extracts the heads of syntactic constructions. It is driven by a set of headrules which are bound to the global list HEADRULES and which record the linguistic facts about heads of English constructions such as the fact that the head of a sentence is the head of the verb phrase which is the verb. Headrules consist of a treefragment as in the templates of the semantic rules, and a "right-hand side" which indicates how to find the head for the node P. If the right-hand side of the headrule is (HEAD n), then the head of the current node is the head of the node which matches node n in the tree fragment; if it is (TERM n) then it is the first word in the terminal string dominated by the node that matches n in the template; otherwise the head is the value of the right-hand side of the headrule (e.g. (QUOTE NEG)). HEAD scans the list HEADRULES for a matching headrule, and if successful, tags the node P with the appropriate head under the tag HEAD.

(HOLD FORM FEATURES)

HOLD is a function for use on arcs of the grammar which adds items which have been found in the sentence in some position

other than their legitimate deep structure position to a special HOLD list. Entries on the HOLD list may later be recognized by VIR arcs in the grammar as if they had been found at the point in the sentence where the VIR arc is applied. The values of FORM and FEATURES are saved on the list so that when the VIR arc is applied, * will be bound to FORM and FEATURES will be bound to the saved value of FEATURES.

(HOLDSCAN HLIST CATEGORY TST)

This function scans the list HLIST (which will be the HOLD list) for elements of the type CATEGORY which meet the condition TST. It is used for processing VIR arcs in the function STEP.

(HYPHENADJ WORD)

HYPHENADJ is a predicate which is true of words which look like hyphenated adjectives. It is a crude approximation of a function to recognize hyphenated strings of English words. HYPHENADJ is true if WORD contains at least one hyphen, at least one alphabetic character, no numeric digits, and no other punctuation marks. It is used in the morphological analysis of adjectives.

(INTERP P TYPEFLAG)

INTERP is the main function of the semantic interpretation component. It computes the interpretation of the node P "as" or "with respect to" the flag TYPEFLAG. That is, TYPEFLAG is a parameter which tells INTERP how to interpret the node P. For example, to interpret a noun phrase as a set instead of the normal

quantification over individuals one can use the typeflag SET. For interpreting normal noun phrases, the three phases of interpreting the determiner structure, the noun itself, and the restrictive modifiers, are indicated by the typeflags NIL, NRULES, and RRULES are used. The NIL typeflag is the normal interpretation which is assumed if no other typeflag is specified. The typeflag TOPIC is used to indicate the interpretation of a node as a Boolean combination of keyphrases.

INTERP's first action is to determine if the node P has already been interpreted with this typeflag, in which case it recovers the interpretation from the TAGLIST and returns without redoing the interpretation. Also for special TYPEFLAG's HEAD, TERM, and IDENTITY, where the interpretation does not require the use of semantic rules, INTERP returns immediately with the appropriate interpretation.

INTERP returns as its value a list of alternative interpretations, called a SEMLIST, which consists of pairs os semantic interpretations and governing quantifiers. Each pair, called an SQ-PAIR consists of a semantic interpretation (SEM) which is to be attached to the current node, and a quantifier (QUANT) which is to be passed up to a governing sentence node.

When the typeflag is not one of the three special cases listed above, the semantic interpreter calls the function RULES with the arguments P and TYPEFLAG to determine the list of semantic rules to use for the interpretation, and calls MATCHER to perform the matching and return the SEMLIST which is to be the value. If there is no semantic interpretation, then DEFAULTSEM may supply a default interpretation (currently only in the case of the typeflag RRULES), but if not, then INTERP either returns NIL or goes into a

break depending on the setting of the flag IHELP.

(JUMP S)

JUMP is a function which can be used on arcs of the grammar to indicate transition to a new state without advancing the input string. It does this by setting up the new configuration CONFIG and returning *L1 to indicate that the function STEP is to continue at location L1.

(LEAFMEMB X LIST)

LEAFMEMB is a function for determining if any of the "leaves" of the list structure X are members of the list LIST.

(LEXIC ALTS)

LEXIC is the function whose job is to determine the next word in the input string. It is called by PARSER whenever the input string is to be moved. LEXIC provides for the expansion of contractions, the substitution of some synonyms, the compactification of compound phrases which are to be treated as single words, and the requesting from the user of definitions for unknown words.

The next word in the sentence is not always uniquely determined, and for this reason, LEXIC is designed to enumerate the alternative possible "next words". It does this as follows: When LEXIC is called, it chooses one of the possible next words and sets up the value LEX to hold it (and adjusts STRING accordingly so that LEX is (CAR STRING). If there are other possible "next words", then LEXIC generates alternatives (ALTCOMP's or ALTSUB's) for these and

returns a list of them as its value. If there is only one possible choice for LEX, then LEXIC returns NIL. The portion of PARSER which calls LEXIC takes any alternatives returned by LEXIC and generates an ALTLEX alternative on the parser's ALTS list. In restarting one of these ALTLEX's, PARSER will call LEXIC with a list of alternatives ALTS, and LEXIC will generate the appropriate choice for another "next word". Thus the first COND in LEXIC tests for whether LEXIC has been called in this mode to enumerate another laternative, and if so branches to location ALT.

Normally, dictionary entries are stored on the property lists of atoms which are provided by the LISP system. However, numbers and the special atom NIL are not permitted to have property lists in LISP nor can pieces of list structure have property lists. However, we would like to be able to recognize such constructions when they occur in input sentences, and therefore LEXIC tests for these special types of LEX. If the input "word" is one of these types, then the functions of the morphological analyzer which are stored on MORPHTESTS can recognize them, and LEXIC will consider them known possible next words. In addition, for pieces of list structure, LEXIC will consider the possibility that the parentheses in the input were superfulous and will generate an ALTCOMP alternative in which the parentheses have been removed. This alternative will not be tried, however, unless there is no other way to parse the sentence.

If the input word is not one of the special forms discussed above, then LEXIC checks to see if the word has a dictionary entry (PLIST determines whether the property list is empty) and, if not, whether the word can be derived by regular inflection from a known word (the function MORPH performs this type of morphological analysis).

If the word turns out to be known for any of these reasons, then the routine branches to location SUBSTITUTE? to consider possible substitutions or compound phrases.

If the "word" is not known, one possible reason is that it contains some punctuation marks that were not separated from it by a space or that it has been run together with another word with only punctuation marks separating them. The next thing which LEXIC does is to look for such punctuation by unpacking the characters of the word and processing them to look for punctuation. In BBN LISP, UNPACK is a function which takes an atom and returns a list of its characters, and PACK is the inverse function which takes a list of characters and makes an atom. Since the 1108 LISP system has no such functions, this feature will not be operational in the 1108 version, and punctuation will have to be separated from the words by intervening spaces.

When a word is known to the system by virtue of having a dictionary entry, then LEXIC looks to see whether the dictionary specifies a substitution to be performed. If so, it will find on the property list of LEX the property SUBSTITUTE followed by a list of alternative substitutions. Each substitution is a list (possibly NULL) of words to be inserted in place of the current word in the input string. If there is more than one substitution, then the first one is taken and an ALTSUB alternative is generated for the rest. Following the testing for substitutions, LEXIC checks for the presence in the dictionary of a COMPOUNDS entry which indicates that LEX can begin a compound phrase. The value of the property COMPOUNDS is a search tree for the possible compounds that can begin with LEX, and LEXIC compares this tree with the sequence of words following LEX in the input string. If it finds a match, it chooses the longest

one and generates ALTCOMP alternatives for any shorter ones.

(LIFTR REG FORM WHERE)

LIFTR is a function for setting register contents at higher levels on the stack. REG is the name of the register, FORM is the value to which it is to be set, and WHERE is a specification of the level on the stack at which the register is to be set. WHERE permits the same options for LIFTR that it does for GETR with the exception of "NEAREST".

(LOG N FILE)

LOG is a function which prints out a record of the sentence processing to a file. N is the number of phases of the processing to be printed (1. parsing, 2. interpretation, and 3. execution). It is called by QGO when LOGFLAG is set.

(LONGBLOCK)

LONGBLOCK is a function which is called by ASSIST to determine the blocked configuration which got the farthest through the input string when a sentence is unparsable. This is a likely site for the error which caused the sentence not to parse, especially if the error has to do with the dictionary entry for a word.

(MARKER X Y)

MARKER is a function which checks whether the word X has the semantic marker Y.

(MATCHER RULELIST P MODE)

MATCHER is the function which matches semantic rules against nodes in the tree. RULELIST is the list of rules to be matched, P is the node to be matched against, and MODE is a flag which indicates what to do with multiple matches. If MODE is AND then multiple matches are ANDed together, IF it is OR, then they are OR'ed, if is is SPLIT, then they are split into distinct (semantically ambiguous) interpretations, and if it is FAIL, then multiple matches cause an error. MATCHER accumulates a list SEMLIST of possible interpretations (S-Q pairs), calling the function MATCHGROUP for each (non-null) element of RULELIST. NIL's in RULELIST serve as "barriers" which terminate the testing of rules if a matching interpretation has already been found in the list, but allow the testing to continue if there have been no matches yet.

(MATCHGROUP RGROUP)

The elements of the list RULELIST in MATCHER may be either single semantic rules or "groups" of semantic rules which are grouped together with a mode operator which specifies how simultaneous matches of different rules are to be handled within that group. MATCHGROUP is the function which handles the matching of such a group. If RGROUP is an atom, then it is a rule to be matched; otherwise it is a group whose first element (like MODE) specifies that simultaneously matching rules are to be SPLIT into different interpretations, AND'ed, OR'ed, or cause FAILure. The first element of the RGROUP is saved on CONJ, and all of the rules in the group are tried. The WHILE expression eliminates the results of non-matching rules, and if there are not more thane one, then the result of the matching rule is returned. In general, each

rule in the group may have returned several distinct interpretations, and the function COMBINATIONS takes all combinations of these. The function SEMCONJ performs the task of combining these interpretations with the operator CONJ.

(MEM N MARKERV)

MEM is the function used in the left-hand sides of semantic rules for asking whether a numbered node in a tree fragment matches a node in the semantic class MARKERV. It does so by verifying that the head of the construction has the semantic marker MARKERV in its dictionary entry, or if the construction is a conjunction that every conjunct is so marked.

(MODAL)

MODAL is a predicate for use on the arcs of the grammar and is true if the current value of * is a modal verb.

(MODESET MODE)

MODESET is a function for initializing some standard mode settings. MODE T causes all parsings to be obtained and interpreted and executed. MODE 1 does parsing only, MODE 2 does parsing and semantic interpretation, and MODE 3 does parsing, semantic interpretation, and execution (i.e. Boolean request generation). MODE DDC sets it up for the standard DDC running option, and MODE NONSTOP sets it up to avoid all interrupts which require console interaction.

(MORPH LEX CATEGORY CMODE)

MORPH is the function which performs morphological analysis for regularly inflected words. LEX is the word being MORPH'ed. If CATEGORY is a single category name, then the analysis is performed for that category only; but if CATEGORY is NIL, then the analysis is performed for all possible categories. MORPH makes use of two tables -- MORPHTESTS and MORPHTABLE. The first contains arbitrary LISP tests for particular types of words, while the second contains inflectional endings.

MORPHTESTS is used both in MORPH and in CATCHECK; it consists of a list of entries for different syntactic categories, with each entry consisting of the name of the category and a series of two-element lists which specify a predicate to be tested and a form to be returned as the form-features list if the predicate is true. A simple MORPHTESTS entry would be (INTEGER ((NUMBERP *) (LIST *))), indicating that any word * which passes the test (NUMBER *) will be considered as an instance of the syntactic category INTEGER, with a standard (root) form identical to itself (*) and with no inflectional features. CMODE is a flag which can be set to skip the MORPHTESTS analysis.

MORPHTABLE indicates the possible inflectional endings for regularly inflected words, and the procedures for obtaining the underlying root forms for inflected words. MORPHTABLE also contains entries for several different syntactic categories. Each entry specifies a syntactic category and then a sequence of entries of the form:

(E- E+ CATEGORY CONDITION FEATURES*)

where E- is an ending to remove from the end of the word (if it is not there, then the rule doesn't apply), E+ is an ending to add to the stem that results from subtracting E-, CATEGORY is the syntactic category of the root which is to be checked, CONDITION is a condition which must be true of the root when viewed as category CATEGORY; and FEATURES (there may be any number of them) are the inflectional features which are to be associated with the word if the condition is satisfied. The CONDITION in the rules is used to verify that the tentative root is indeed in the class of words which undergo the regular inflection represented by the rule. For example, the entry (N ((S) NIL N (PLURAL -S) (NUMBER PL))) says that if we are looking for a noun (N) and if the word ends in S, then we remove the S from the end, add nothing (NIL) and look at the resulting word as a noun (N) to test the condition (PLURAL -S) (which tests the dictionary entry for the word for the property N with value -S) to see if the word undergoes this type of inflection. If so, then the inflectional features associated with the word consist of the single feature (NUMBER PL).

If a dictionary entry is computed for a word by means of morphological analysis, then it is added to the property list for that word for the duration of the console session with the system. Thus, the morphological analysis described will be done only once for each word for which it is required.

(MORPHTABCHECK TABLE)

MORPHTABCHECK is the function which tests the entries in the MORPHTABLE for MORPH. It returns a list of appropriate form-features lists for the dictionary entry if a line of the table is successful and NIL otherwise.

(MORPHTSTCHECK TAB)

MORPHTSTCHECK is the function which checks entries in the MORPHTESTS table for MORPH. It also returns a list of form-features lists.

(NEGSORT CLAUSE)

NEGSORT is a function used in the Boolean request generation to move negated phrases to the end and convert them to calls to SDIFF, the function which indicates the set difference between two Boolean descriptions. See BOOLREQ for more details.

(NEXTWRD WRDS)

NEXTWRD is a condition which is used on arcs of the grammar for looking ahead one word in the input string. If WRDS is a single word, it compares it with the next word in the sentence, or if WRDS is a list of words it checks to see if the next word in the sentence is a member of the list.

(NOR . ARGS)

NOR is a logical operator for NOT OR.

(NPBUILD)

NPBUILD is the function which builds the syntactic tree structures for noun phrases. It is used on POP arcs in the grammar.

(NPCHECK NODE TERMINALS)

NPCHECK is a function used in PNCHECK for testing constituents of a noun phrase node. It uses the free variable NP which is bound to a noun phrase node by PBCHECK and looks for a constituent of the noun phrase of type NODE. It checks whether the immediate constituent of this NODE is a member of the list TERMINALS.

(NPR X)

NPR is a variant of QUOTE which is sometimes used in semantic rules.

(NULLR REG)

NULLR is a predicate for use in conditions in the arcs of the grammar for testing whether the register REG is empty.

(NXTVAR)

NXTVAR is the function which gets the next available variable name for use in the quantifiers during the semantic interpretation. It uses variables cyclically from a list called VARIABLES.

(ORFLAG X)

Because of the tendency for people to say "and" when they "really" mean "or" in requests to the system, ORFLAG provides a way to instruct the semantic interpreter to interpret

the user's "and"'s to use "or", and he can call ORFLAG(NIL) to restore the original status. This allows the user to adjust the system to meet his own needs. If he can comfortably adjust to thinking of the consequences of using "and" in a request and phrase his requests appropriately, then he can operate in the system with the normal setting, but if he finds that he habitually says "and" when he means "or", then he can set ORFLAG(T) to alter the system's behavior to suit his own.

(ORMATCH TEMPLIST)

ORMATCH is a routine for matching OR'ed templates in RMATCH. That is, when in place of a single template in a semantic rule, there is an OR of several templates, then ORMATCH is called to perform the matching of all of them. It also provides for a standard DEFAULT interpretation as the last component of an OR. It will take the default interpretation if and only if there are no other matching templates in the OR.

(PARSEPARENS PARENSTRING START)

PARSEPARENS is a function for use on the arcs of the grammar for parsing parenthetical expressions. It allows the grammar to call the parser recursively on the parenthetical string PARENSTRING with the specified start state START.

(PARSER STRING MODE ALTS)

PARSER is the controlling routine of the parsing component. STRING is the sentence to be parsed and MODE is a variable which governs the mode in which the parsing is to

proceed. (ALL causes all parsings to be found, SPLIT causes all parsings to be followed in parallel, and non-null values in general cause automatic selection of a new alternative whenever a blocked configuration is encountered.) ALTS is a list of alternatives which is NIL unless PARSER is being called to continue looking for parsings, in which case it will be list of alternatives generated by a previous call to PARSER. PARSER returns a list whos first element is a list of parsings found, and whose second element is a list of alternatives which it did not try. It is this list of alternatives which can be used to continue looking for additional parsings if the first one is found not to be satisfactory.

PARSER manages a list of active configurations (ACFS) which it calls the function STEP to advance. A configuration consists of a complete record of a state of the machine -- i.e., a list of the state, stack, registers, contents of the HOLD list, and a path entry which records the history of how the configuration was reached from the initial configuration. PARSER runs in two modes depending on the setting of a flag LEXMODE. In the normal mode, LEXMODE is NIL and the parser proceeds by calling LEXIC to determine the next word in the string. LEXMODE is set when the parser is operating on a reduced conjunction during the part of the processing when the suspended configuration for the first conjunct is being resumed on a tail of the string consumed by the second conjunct (See SYSCONJ and CONJOIN). At this time, the parser follows the trail (TRAIL) left by the previous parsing of this substring, and the normal lexical analysis is bypassed. This is due to the fact that the two components of the conjunction are required to analyze the shared substring in the same way.

If after calling LEXIC, the current word LEX is still an unknown word, then the configuration is added to a list of blocked configurations and the parsing is aborted under the assumption that no other alternatives will be able to parse beyond the unknown word in the sentence. When LEX is a known word, however, PARSER calls the function STEP to advance the active configurations and produce a new list of active configurations at the next position of the input string, it advances the input string to the next position and repeats. If at any time there are no new active configurations, then depending on the setting of the flag MODE (which is normally set to the value of the global flag PMODE) it either goes into a break at location HELP, or it selects an alternative to be tried by calling the function DETOUR at location ALT. If there are no more alternatives, but there have been some complete parsings found (if so they are stored on VALUES by the function POP which is executed in interpreting the POP arcs in the grammar), then PARSER returns those parsings. If this call to parser was not itself an attempt to find additional parsings (in which case ALTFLAG would be set), then the failure to find any parsings of the sentence will cause a call to ASSIST. This is the function which would eventually contain facilities for making helpful diagnostic comments to the user as to the likely cause of the error, and perhaps even correct them and continue. At the moment it merely goes into a break (if the flag AHELP is set) at the blocked configuration which got the farthest into the string before blocking.

The various locations ALTCONJ, ALTLEX, and ALTARC know how to restart their corresponding types of alternatives, which have been found on the ALTS list by DETOUR.

(PLURAL ENDING)

PLURAL is a function for use as a condition in the MORPHTABLE of the morphological analysis component. It tests whether the dictionary entry for the current word * is marked as a noun with regular inflection of the type ENDING.

(PNCHECK NP PNCODE)

PNCHECK (person-number check) is the function which checks for person-number agreement between a noun phrase (NP) and a person-number code (PNCODE). It is used in the grammar to check person-number agreement between verbs and their subjects.

(POP POPVAL POPFEATURES)

POP is the function which returns from a recursive call in the transition network grammar. It is used by the function STEP for the interpretation of POP arcs, and can occasionally be used as an action on an arc of the grammar. POPVAL is the structure that is to be returned from the recursive call (and bound to the current constituent pointer *) and POPFEATURES is the list of features which is to be associated with the current constituent.

POP restores the configuration which was saved on the stack at the time of the PUSH which initiated present level of computation and performs the actions on the push arc, after setting the flag NOMOVEFLAG which indicates that the function TO at the end of the PUSH arc is not to advance the input string (since it has already been advanced by the recursive computation).

If the stack is empty, and the STRING is also empty, then POPVAL is a complete parsing of the sentence, and is added to the list VALUES which is being maintained by PARSER. If the string is not empty at this time, then the configuration is blocked.

(POPCONJ)

POPCONJ is one of the functions used for the facility which handles reduced conjunctions (see CONJOIN and SYSCONJ). A call to POPCONJ is placed on the stack by CONJOIN when resumes the suspended configuration for the first conjunct in a conjunction. This call to POPCONJ will be invoked when the first conjunct has been completed, at which time it will determine whether the two components of the conjunction are compatible, compute the syntactic representation of the conjoined phrase, set up a configuration on the alternatives list for the computation which is to be resumed at this point, and abdicate control by returning *END as its value. This will enable DETOUR to pick up the configuration from the ALTS list and continue parsing. (This method of proceeding with the parsing is used to restore the value of the input string which has been temporarily destroyed by the operation of STEP in LEXMODE mode without interfering with any other active configurations which may be on the current ACFS list.).

(PPATH:BACK PPATH)

PPATH:BACK is a function for backing up along the path entries for a configuration. It's argument is a partial path (PPATH) which consists of a record of an augmented configuration

(ACONFIG) and the arc which was followed from that configuration. It lacks the information about the computation of that arc which is a part of a complete path. (See the listing FORMATS in the computer listing for the specification of the LISP structure formats for partial paths, paths, configurations, and augmented configurations.) If the partial path PPATH is not the first one in a call to the network, then its last element is the full PATH entry recording the configuration prior to the current one, and CDR of this is the partial path associated with it. If there is no previous path as the last element of PPATH, then this configuration is the first one after some PUSH (or indeed the first one in the analysis of the string) and the preceeding partial path is taken from the STACK associated with PPATH.

(PPATH:HOLD PPATH)

This function extracts the HOLD list from a partial path.

(PPATH:PATH LIST)

This function extracts the previous path entry from a partial path.

(PPATH:REGS PPATH)

This function extracts the registers list from a partial path.

(PPATH:STACK PPATH)

This function extracts the stack from a partial path.

(PPT XTR FILE)

PPT (pretty print tree) prints a parse tree (XTR) in a pretty format to the file FILE. The function which actually does the printing is PPT1.

(PPT1 XTR KID FILE)

See PPT.

(PRED SEMFORM)

PRED is one of two functions (PRED and QUANT) which are used in the right-hand sides of semantic rules to indicate what is to happen to quantifiers. QUANT indicates that the right-hand side is a quantifier that is to be passed up to a higher constituent, with the semantic interpretation of the current node being the variable assigned to that quantifier. PRED indicates that the right-hand side is a predicate which is to "grab" any quantifiers passed up by constituents -- that is any such quantifiers will be treated as quantifying the expression SEMFORM which is the argument to PRED.

(PRINTOUT *X*)

PRINTOUT is the function used to printout answers in the retrieval component. It increments the free variable COUNT which is maintained by EXECUTE.

(PRINTPARSES FILE)

PRINTPARSES is the function used by SENTPROC to printout the result of the parsing when the appropriate flags are set. If the global flag PPTFLAG is set, then this happens using PPT to obtain the printout in a pretty format. Otherwise, the printing is in the ordinary parenthesis notation corresponding to the internal list structure.

(PUSH PS)

PUSH is the function used by STEP to interpret PUSH arcs in the grammar. It can also be used as an action on the arcs of the grammar under certain circumstances. In the normal mode (when LEXMODE is not set) it saves the current state, register contents, actions to be performed, HOLD list, and partial path on the stack and starts a new configuration at the lower level with the initial register contents from SREGS (those register contents sent down by calls to SENDR). When LEXMODE is set, then PUSH must take its constituent from the trail which the parser is following. If nothing was sent down with a SENDR, then it merely takes the value stored in TRAILVAL for the trail being followed. If there were register contents sent down, however, then it calls REDO to follow the path associated with the computation of that constituent to construct the new constituent based on the new initial registers sent down.

(Q . QUERY)

Q is the function called by TALKER for processing input sentences. It sets the variable SENTENCE, calls SENTPROC, and logs the resulting output if LOGFLAG is set.

(QGO LABEL)

QGO is the function called by TALKER to continue looking for more parsings, to repeat a semantic interpretation, etc. It calls SENTPROC with a label LABEL which specifies a location within SENTPROC at which processing is to be started.

(QSTART)

QSTART is a predicate used in the grammar at the beginning of a sentence to determine whether it looks like a question -- i.e., it starts with an interrogative word or with an auxilliary verb.

(QUANT SEMFORM)

QUANT is a function used in the right-hand side of semantic rules to indicate that SEMFORM is to be interpreted as a quantifier. (See PRED.)

(REDO TRAIL REGS)

REDO is a function called by PUSH when it is following a trail during the LEXMODE phase of the recognition of a reduced conjunction. It will redo the computation indicated by TRAIL starting with the register contents REGS instead of those which were originally used in TRAIL.

(REFLOC RHSFRAG RVECTOR)

REFLOC is a function used by SEMSUB in the semantic interpreter to make up REFLISTS for a given right-hand side of a rule and a given vector of matches (RVECTOR). If RHSFRAG (a fragment of the right-hand side of the rule) is a REF (i.e., an expression which refers to the semantic interpretation of some constituent of the node being interpreted), then REFLOC returns the REFLIST for that constituent. Otherwise it scans RHSFRAG for instances of REF's. The REFLIST which is returned consists of the pointer (SUBP) to the constituent in the tree to which the REF refers (for this particular match specified by RVECTOR), the REF itself (RHSFRAG), and the interpretation of the node SUBP using the typeflag specified by the REF.

(REFP RHSFRAG)

REFP is a predicate which tests a fragment of a right-hand side of a semantic rule to determine whether it is a REF (i.e., whether it refers to a constituent of the tree being interpreted whose semantic interpretation is to be used as a part of the current interpretation). This is true if the fragment is either a dotted pair of integers, or a list beginning with the atom #.

(REFPTR RHSFRAG)

REFPTR is like REFP, except that it also returns a pointer to the node in the tree to which the REF refers (the matching pointer being obtained from the current RVECTOR).

(REFQUANTS REFVECTOR)

REFQUANTS is a function which gathers the quantifiers from all of the REFLIST's in REFVECTOR into a single quantifier "collar" which is to be wrapped around the expression which it governs. It assumes that the REFLIST's on REFVECTOR have been sorted into the order in which they are to occur. This sorting is accomplished by the function SORTREFS in a call from SEMSUB.

(REFSUB REFVECTOR)

REFSUB is a function which takes the current value of RHS (maintained by SEMSUB), substitutes the semantic interpretations of its REF's, and evaluates the result to obtain the semantic interpretation of the current node. REFSUB1 actually performs the substitution, and prior to the execution of the substituted right-hand side, REFQUANTS is used to construct the appropriate quantifier "collar". The call to EVAL will result in these quantifiers being "grabbed" and wrapped around the semantic interpretation of the current node if the right-hand side of the rule (RHS) is embedded in a PRED; if it is embedded in a QUANT, then the call to EVAL will result in the quantifier being inserted into the "hole" of the collar (substituted for DLT) and the semantic interpretation of the current node will be set to the variable name associated with the quantifier.

(REFSUB1 RHSFRAG REFVECTOR)

REFSUB1 performs the substitutions in the RHS of a semantic rule before it is evaluated by REFSUB. It substitutes the current value of the variable (QVAR) for occurrences of the atom "X" when interpreting restrictions on the range of quantification in interpreting noun phrases, and substitutes the semantic interpretations for REF's.

(REFSUB2 RHSFRAG REFVECTOR)

REFSUB2 is used by REFSUB1 to walk across a sublist of a RHS and apply REFSUB1 recursively.

(REFTYPE REF)

REFTYPE is a function for extracting the reftype of a REF -- i.e. the typeflag that is to be used for interpreting the node to which the REF refers. For dotted pairs, the REFTYPE is NIL, while for REF's that begin with #, the reftype is the element of the list which follows the numbers that denote the node to be interpreted.

(RELTAG PLIST)

RELTAG is the function used by the semantic interpreter for locating the relative pronoun of a relative clause to be interpreted and tagging that node with the variable of quantification (QVAR) associated with the noun phrase which the relative clause modifies.

(REQUESTDEF LEX)

REQUESTDEF is the function which is called to interact with the user of the system when an unknown word is encountered by the parser. It prints out a comment followed by the unknown word, and goes into a break to allow the user to define the word (using DDEF) or to change it (using CHANGEWORD).

(RESUME TEMP)

RESUME is a function which can be called on an arc of the grammar to resume a PUSH computation which has been assigned a feature RESUME by the function RESUMTAC. This provides for the termination of a PUSH computation at one point in the string and resuming it later at another part of the string, and it provides a mechanism for handling certain phenomena which would be called right-extraposition transformations in transformational grammar theory.

(RESUMETAG STATE)

RESUMTAG is a function for computing a RESUME feature for a configuration which will enable it to be resumed later beginning in state STATE. It is used by arcs of the grammar in conjunction with the function RESUME.

(RFEAT . ARGS)

REFEAT is a function for retrieving syntactic features from the dictionary entries for words. ARGS is a list whose first element is the mane of the syntactic feature desired, and whose second element indicates the word whose dictionary entry is to be consulted (which may be indicated either by the name of a register which contains it, by the special pointer *, or by some other LISP expression).

(RMATCH RULE P MODE)

RMATCH is the basic semantic rule matching function. It matches the single semantic rule RULE against the node P with mode MODE, (unless MODE is reset by the first element of the rule itself). It calls TEMPMATCH to match each of the templates of the rule or

ORMATCH to match OR'ed groups of templates, and if a successful match is found it calls SEMSUB for each possible way in which the rule can match. If there are multiple matches, then it combines them in the way indicated by MODE.

(RULES P TYPEFLAG)

RULES is the function called by INTERP to decide what semantic rules to use to interpret the node P. This depends both on the type of node P and on the TYPEFLAG. Rules are taken either from global lists (DRULES, PRERULES, etc.) or from the property lists of the head of the construction P (SRULES, RRULES, NRULES). RULES also sets MODE to the value which is the normal default for the type of node being interpreted.

(SBUILD)

SBUILD is the function called by the grammar for building the syntactic structures of sentences. It gathers up the various pieces of the structure from the registers in which they have been stored and assembles them into a syntactic tree using the function BUILDQ.

(SCANSTACK TEST)

SCANSTACK is the function which scans the stack looking for a stack level which satisfies the test TEST. It is used in LIFTR and GETR for locating levels of the stack where registers are to be set or interrogated.

(SCOMP V)

SCOMP is a function which tests whether a verb V takes a sentence complement by checking the dictionary entry for V.

(SEMCONJ CONJ SEMLIST)

SEMCONJ is the function which combines multiple semantic interpretations with the conjunction CONJ. It conjoins the SEM's of the interpretations under the conjunction CONJ, and produces a quantifier which is the nesting of all of the quantifiers which are associated with the individual interpretations.

(SEMNET N1 N2)

SEMNET is a function which can be used in the grammar for testing the compatibility between a noun N1 and a noun N2 to determine if the first can modify the second as a noun-noun modifier. Various conditions have been tested for this purpose, but in the absence of semantic information or access to the thesaurus of the retrieval component, the function currently is unconditionally true.

(SEMSUB RHS RVECTOR)

SEMSUB is the function which substitutes semantic interpretations for their REF's in the right-hand sides of semantic rules. It is the major dispatcher among the functions REFLOC, SORTREFS, and REFSUB.

(SENDACTP ACTION)

SENDACTP is a predicate used by STEP and REDO for identifying actions which send register contents down to lower levels (i.e. SENDR and SENDRQ).

(SENDR REG FORM)

SENDR is a function which sets the contents of the register REG to the value of FORM at the next lower level to which control will be passed by a PUSH arc.

(SENDRQ REG FORM)

SENDRQ is like SENDR except that FORM is not evaluated.

(SENTPROC SENTENCE LABEL)

SENTPROC is the major dispatching routine for the processing of an input sentence. It dispatches the input to the various routines PARSER, SPROC, and EXECUTE, times computations, prints out intermediate results and timings, and logs the results as appropriate. It also provides for the feedback to the parser to obtain additional parsings if the semantic interpretation of the first parsing fails (up to maximum number of times specified by MAXREPARSES), and for the redoing of a previous execution or interpretation or the continuation of parsing by calls which specify a LABEL of EXECUTE, INTERP, or PARSE, respectively.

(SETR REG FORM)

SETR is the function which sets the contents of a regisete REG to the value of form at the current level of processing.

(SETRE REG FORM)

SETRE is like SETR except that REG is evaluated to obtain the name of the register to be set.

(SETRQ REG FORM)

SETRQ is like SETR except that FORM is not evaluated, but taken literally.

(SHOWTIME CONSES TIME FILE)

SHOWTIME is the function which prints timing information to a file in the BBN LISP system. In the 1108 system, since the timing facility will not be available, SHOWTIME doesn't do anything.

(SORTREFS REFLISTS P)

SORTREFS is the function which sorts REFLISTS into the order in which the quantifiers associated with the REF's are to be incorporated into the interpretation -- namely in order of their left-to-right position in the structure P. This is accomplished by the function SORTREFS1 which walks the tree P and adds REF's to the list in the order in which it sees them.

(SORTREFS1 REFLISTS P)

See SORTREFS.

(SPLIT . SPLITARCS)

SPLIT is a function which can be used on arcs of the grammar to cause two or more alternatives to be followed at once. SPLITARCS is a list of alternative continuations of the arc on which the SPLIT action occurs, and all such alternatives will be followed in parallel. This feature has not been used in the current DDC grammar.

(SPROC P)

SPROC is the function which begins the semantic interpretation of the node p and returns the list of possible semantic interpretations. It calls the function INTERP which does the work.

(STACKELT:PPATH LIST)

This function is used to extract the PPATH entry from an element of the stack.

(STEP CONFIG ALT)

STEP is the major function of the transition network parser. Its job is to take a single configuration (CONFIG) from the active configurations list of PARSER and compute from it a list of configurations which are possible at the next point in the input string. It takes the list of arcs for the state of the configuration and considers each in turn until it finds one which can be followed. It also interprets the conditions and actions on the arcs, and generates ALTARC alternatives on the ALTS list for any arcs which remain untried when it decides to follow one.

STEP is also the function which is called to pick up the processing of an alternative taken from the ALTS list. In this case, the argument ALT will be set to the alternative to be restarted, and the setting of CONFIG will be irrelevant. In this case, STEP will branch to location ALT where it determines the type of alternative and does the appropriate thing to resume the processing.

At location LØ, STEP unpacks the configuration CONFIG into its component parts (STATE, STACK, REGS, HLIST, and PATH), and at location L1, it begins the determination of the list of arcs to be considered. If, however, the time already spent in the parsing exceeds a global limit MAXTIME, the parsing is terminated with an appropriate comment. If the current LEX (the current word in the string) is marked with the property LEXARCS, then it is an "interrupt word" and the list of arcs to be tried is not taken from the value of the state name as would usually be the case, but is instead computed by the expression which is the value of the property LEXARCS. This facility allows for the convenient handling of special function words which can occur at almost any point in a sentence with a regular effect. For example, the conjunction scope indicators "both" and "either" are handeled by this facility in the current system. Another special case for the determination of a list of arcs other than that listed for the state is the SYSCONJ facility. If the flag SYSCONJFLAG is set and the current LEX is a conjunction and there there are no CAT CONJ arcs leaving the current state, then the SYSCONJ facility provides its own special default CAT CONJ arc in place of the normal list of arcs. This does not happen when LEXMODE is set, however (i.e. when STEP is already interpreting a part of a reduced conjunction). When the global flag SPLIT is set, the list of arcs will be moved to a list of "untried" split alternatives (SPLITS) and control will branch to END where there is a test for uncompleted SPLITS (i.e. alternatives to be followed in parallel) before returning. Normally, however, the list of arcs is taken

from the value of the state, and control passes to L2.

L2 begins the basic loop which tries successive arcs from the list ARCS. If there are no more arcs, then depending on the settings of various mode variables and other parameters, control either passes to END or HELP. Also if MODE is non null, the blocked configuration is added to the list BLOCKS (for later use by LONGBLOCK in ASSISTER), if the number of blocked configurations exceeds the global parameter MAXBLOCKS, the parsing is terminated.

L3 begins the processing of the arc selected by initializing the values of *, FEATURES, SREGS, and NOMOVEFLAG. The atom * is the pointer to the current constituent (initially it is equal to the current word LEX, but after popping from a lower level it is the value of the constituent returned, and on a virtual ARC it is the value of the constituent which is taken from the HOLD list). FEATURES is the list of features associated with the current value of *, and SREGS is the list of registers which have been sent down to the lower level by SENDR actions (it will only be non-null after the execution of SENDR actions immediately prior to a PUSH to a lower level). NOMOVEFLAG is a flag which indicates whether the input string is to be advanced after the transition caused by the arc (it is initially set to NIL indicating that the string is to be advanced, but various actions on the arc can cause it to be reset). The major part of the function STEP consists of the SELECTQ at location L3 which determines the type of arc and performs the appropriate actions.

A CAT arc is followed if LEX can be a member of the syntactic category indicated by the label on the arc (ARC:LABEL ARC). If LEXMODE is set, however, this arc can only be taken if the word was also taken as a member of this category in the trail which is

L3

being followed. The value TEMP which is set by the call to CATCHECK or taken from the trail (TRAILVAL) is a form-features list whose CAR is the root form of the word LEX and whose CDR is a list of inflectional features for the word. These values are bound to * and FEATURES, respectively, as a result of choosing a CAT arc, and control passes to location TST which checks the conditions associated with the arc.

A PUSH arc indicates a recursive application of the network to find a phrase of the type recognized by the state which is given as the arc label. The condition on the arc is tested before the PUSH in order to determine whether to perform the PUSH. If LEXMODE is set, then the PUSH does not occur unless the corresponding entry on the trail being followed was also a PUSH to the same state. To facilitate the use of SENDR's to send register contents down into the lower level prior to the push, there is an optional constituent of the PUSH arc immediately after the condition on the arc which consists of a list of actions to be executed prior to the actual call to PUSH. This list of actions is indicated by an initial element "!" (which will print "!!" in the 1108 system). Also for the same reason any initial sequence of actions of the SENDR type (tested by SENDACTP) are executed prior to the PUSH. The call to the function PUSH will save the current state and register contents and the uncompleted actions on the arc on the pushdown stack for continuation after the embedded phrase has been recognized.

POP is a "pseudo" arc in the sense that it has no "destination" at the end. Rather it indicates a return from an embedded computation to the configuration which PUSH'ed for it. It is represented as an arc so that its choice can be ordered with respect to those of the other arcs and so that it can be made conditional on the context by using a test on the arc. POP arcs are not

permitted when LEXMODE is set, since the PUSH's are never actually executed in this case (and therefore, the trails which are being followed never have POP arcs on them). POP's are also forbidden if there are entries on the HOLD list put on at this level which have not yet been used by any virtual (VIR) arc (this is part of the HOLD facility for dealing with left-extradeposition transformations). SPOP is a variant of POP which is used in some systems for selective modifier placement, but is equivalent to POP in the current system.

A JUMP arc is an arc which performs some actions but does not advance the input string. The label on the arc names the state to which control is to go after the actions are executed if there is no terminating action on the arc.

A VIR (virtual) arc is an arc which picks up a constituent from the HOLD list (placed there by a call to HOLD on some arc of the grammar) and treats it as if it had just pushed for and found the constituent at this point in the string. It sets * and FEATURES to the values taken from the HOLD list and then executes the actions on the arc (after setting NOMOVEFLAG to prevent the input string from advancing).

A WRD arc tests for the presence of a particular word in the input string. Similarly a MEM arc tests for one of a specific list of words. A TST arc allows for the testing of an arbitrary condition expressed in LISP as the condition on the arc. The label on a TST arc has no effect on the operation and can be used for purely mnemonic purposes by the grammar writer.

A SUSPEND arc is an arc which suspends the processing of the current state with an incremented weight (incremented by the amount indicated in the arc label). This can be used to control the order in which parsings are discovered by suspending "unlikely" alternatives to be tried only after more likely possibilities have been tried. There is also a SUSPEND action which can be used on an arc to suspend the processing of just that arc.

A SPLIT arc is essentially a group of arcs grouped together with the "conjunction" SPLIT to indicate that the arcs in that group are to be followed in parallel. It is similar to the SPLIT action which can be used on arcs to indicate parallel alternative "tails" for a single arc.

A DO arc is an unconditional list of actions to be performed with a destination specified at the end.

The location TST performs the checking of conditions on the arcs for a number of different arc types, and similarly the location ACT executed the actions on the arcs. The location ALT performs the appropriate actions for resuming an alternative, and HELP provides a break for user interaction in certain cases.

The location END is entered when a given configuration either blocks or is completed. It checks whether there are any uncompleted configurations (UCFS) placed there by SPLIT actions on arcs, and if so processes them. It tests also for any unprocessed configurations (SPLITS) placed there by a SPLIT arc or by the mode flag MODE being set to SPLIT, and it processes all of these before returning. When all "parallel" computations have been performed, it returns the list (VCFS) of resulting configurations which have been constructed. (The actual construction of the resulting configurations is performed by the function TO when it occurs as terminal action on an arc.)

46

(STORALT ALT)

STORALT is the function used in many places for placing alternatives on the ALTS list.

(SUPFORM ENDING)

SUPFORM is a predicate for use in MORPHTABLE entries for testing the type of conjugation which an adjective undergoes for the superlative form.

(SUSPEND N)

SUSPEND is an action for use on arcs of the grammar for suspending a given computation in favor of "more likely" ones. It increments the weight associated with the current computation by the amount N and generates an ALTARC alternative on the ALTS list.

(SYSCONJ STATES)

SYSCONJ is the action which invokes the system conjunction (SYSCONJ) facility for reduced conjunctions. It can either be used on CAT CONJ arcs by the grammar writer, or it will be supplied automatically for states which don't have CAT CONJ arcs if SYSCONJ-FLAG is on. It is the first function of the trilogy of SYSCONJ functions (SYSCONJ, CONJOIN, AND POPCONJ) to be executed. It causes the insertion of a special stack entry with a call to CONJOIN into the stacks of a set of restart configurations (computed by CONJSTARTS) and the generation of an ALTCONJ alternative for each such configuration. It then returns *END so that STEP will terminate the current configuration and pick up one of the generated ALTCONJ alternatives.

(T:REF NODE)

T:REF is the function which assigns to nodes in the syntax tree a reference which is used for associating information with that node in the TAGLIST. It is currently identical with the LISP pointed to the node.

(T:SONS NODE)

T:SONS is a function for computing the list of the sons of a node of a tree. It depends on the notation being used for trees in the system. If it is the two-paren notation, then the sons are the CADR of the node; otherwise they are the CDR.

(TAG P TAGNAME VALUE)

TAG is the function which places tags on the TAGLIST. It associates with the node P the property TAGNAME with the value VALUE.

(TAILS LIST)

TAILS is a function for enumerating the tails of a list. (E.g. the tails of (A B C) are (A B C), (B C), and (C).) It is used by CONJOIN for computing the possible tails on which the suspended first conjunct of a conjunction may be resumed.

(TAILS1 LIST)

TAILS1 is like TAILS but it omits the singleton tail. (E.g. TAILS1 of (A B C) gives (A B C) and (B C), but not (C).)

48

(TALKER MODE)

TALKER is the major executive of the DDC English Language preprocessor. The first thing to be done by a user after loading the system is to call TALKER with an argument MODE (usually NIL) to indicate the mode in which he wants to operate. (MODE of NIL indicates use of the mode settings as they exist at that time without change, while a non-null MODE will cause MODESET to be called to set the appropriate mode variables.) TALKER takes care of the interaction with the user, accepting sentences and LISP commands as input, and performing the appropriate actions for each. In the BBN LISP system it also takes care of saving the input sentences on a history list which enables the user to refer to and reuse the results of his previous typein, but in the 1108 version, these facilities will be inoperable and the functions which call them will be "no ops" (i.e. will cause no operations). TALKER also takes care of keeping a record of the imput sentences typed when the flag INPUTLOGFLAG is set; this facility allows for the monitoring of user activity to obtain statistics for improving the performance of the system.

(TEMPMATCH TEMPLATE P)

TEMPMATCH is the function which matches templates with nodes of the tree in the semantic interpreter (it is called by RMATCH and ORMATCH). It calls the functions TMATCH to perform the tree matching of the tree fragment with the node P, and calls CCHECK to check to semantic conditions of the template for any resulting tree matches. It also provides the results of a simulated match in the case of a DEFAULT template.

(TERM TREELIST)

TERM is a function which returns the list of "leaves" or "terminal nodes" of a list of tree structure nodes (TREELIST). It does so by walking the tree structures and gathering up the "leaves".

(TIMEP X)

TIMEP is a function which can be called for morphological analysis of an atom which looks like a time (i.e. a number less than 24 followed by a colon follwed by a number less than 60).

(TMATCH PLIST FLIST)

TMATCH is the function which performs the subtree matching for the semantic interpreter (called by TEMPMATCH). PLIST is a list of nodes not the tree which are to be matched against the fragment nodes in the list FLIST. It returns a list of all possible matches--each match being represented by a vector (ALIST) of correspondences between numbered nodes in the tree fragment and nodes in the tree being interpreted.

(TO S)

TO is the function used by arcs in the grammar to indicate the destination (next state) for an arc. It computes the configuration which results from the transition and returns a label to STEP (through ACT) indicating what location it should pass control to (which depends on such factors as whether NOMOVEFLAG is set, whether it is at the end of the string, etc.)

(TRACER . ARGS)

TRACER is a function which is called at many points in the parser for providing a tracing of the course of the parsing when the flag TRACE is set. It is an extremely valuable tool for debugging grammars, and is also a useful instructional tool for teaching the operation of the parser and the grammar to staff.

(TRAIL PATH)

TRAIL makes a list of the path entries in a path in the order in which the transitions occur so that they can be followed by the parser in LEXMODE mode. (The normal order of entries in a path is reversed and "right nested".)

(TRAIL1 PATH)

TRAIL1 is like trail except that it skips the configurations that occur immediately after JUMP and VIR transitions.

(TRAILS PATH)

TRAILS is a function called by CONJOIN to make a list of the trails at different levels of the analysis of the right-hand shared portion of a reduced conjunction. These trails are the possible trails on which the suspended first component of the conjunction can be resumed.

(VPARTICLE . ARGS)

VPARTICLE is a function for testing whether a verb combines with a particle to form a verb (e.g. "call up," etc.). ARGS is a list whose CAR specifies the verb in question (usually by naming the register which contains it) and whose CADR specifies the particle in question (again by naming a register or by reference to the pointer *).

(VPASSIVE V)

VPASSIVE is the predicate which tests whether a verb V can be passivized. This is true either if it is so marked in the dictionary or (as a default) if it is totally unmarked for syntactic features.

(VTRANS . ARGS)

VTRANS is a function which tests whether a given verb is transitive (i.e. whether it can take a direct object). This is true either if the verb is so marked in the dictionary under the property FEATURES, or (as a default) if the verb is not marked with any syntactic features at all.

(WRD . ARGS)

WRD is a function for use in conditions in the grammar to test whether the current word * or a word in some register is a member of a list of words. CAR of ARGS is the list of words to be tested, and CADR of ARGS specifies the word to be tested.

252

Appendix E.

THE ORGANIZATION OF THE DDC DICTIONARY

The following description of the DDC dictionary is intended to serve two purposes: first, to provide a general picture of the dictionary, indicating what types of information must be specified for lexical entries; and second, to demonstrate the precise format in which this information must be represented. Hopefully, then, this description will enable DDC personnel to develop appropriate definitions for words not included in the set of sample queries.

I. An Overview

The dictionary entry for a given word is stored on its LISP property list as a sequence of property-value pairs (see the appendix for a formal specification of the syntax required in a definition. Usually the properties will be the names of lexical categories (e.g. N, V, ADJ), indicating that the word can be a member of the category, but three other properties are allowed: SUBSTITUTE, COMPOUNDS, and FEATURES. SUBSTITUTE supplies a mechanism for mapping abbreviations and alternative spellings of a word into a single form, which contains the full dictionary entry. If a word can be the first word of an idiom or compound expression (e.g. "United" in "United States"), then the property COMPOUNDS denotes the following word in the compound and a standard form which will replace the whole sequence when it is found in a string. Thus the pair COMPOUNDS ((STATES UNITED-STATES)) on the property list of UNITED would convert all occurrences of the sequence UNITED STATES into the single word UNITED-STATES, which then must be entered separately in the dictionary. The implementation of the lexical category properties, SUBSTITUTE, and COMPOUNDS, all support the general philosophy that the dictionary information for a number of related items should be stored on only one standard form but should be accessible by any of the items.

II. Lexical Categories

The lexical categories are the properties explicitly referenced by the grammar and the parsing algorithm. When the grammar asks if a word is in a particular lexical category, the dictionary look-up routines provide a yes-no answer and, if yes, two kinds of information: (1) the root form of the word, and (2) a set of inflectional features. Thus if the grammar asks if BOOK is a noun, the answer is "yes--with root BOOK and inflectional feature (NU SG)". For the verb TALK the root would be TALK with inflectional features (TNS PRESENT) and (PNCODE 3SG).

The value of the lexical category property encodes the root and feature information in several ways (see Appendix I). The most transparent notation is simply a parenthesized sequence (a list) whose first element is the root and whose succeeding elements are the features. If the word has a number of different interpretations within a single category (e.g. SAW as a verb), the value of the category property is a list of root-feature lists, one for each interpretation. If the value of the property is an atom, (a character-sequence instead of a list), then the root features are supplied by default dictionary routines: If the value is "*", then the word itself, is taken as its own root and the set of features is the empty set. For any other atomic value, the root is still the word itself, but a default set of features is provided, depending on the category (e.g., nouns are marked as singular by default).

Atomic values have another side-effect: they specify the morphological paradigm of which the word is the root. Thus for verbs, the atomic value S-ED indicates that the third-person singular is formed by adding an S, the past tense and past participle result by adding ED, and the present participle is formed with ING. With the inflectional paradigm encoded in this way, only the root forms of regular verbs, nouns, adjectives, and adverbs

must be entered in the dictionary. Definitions for inflected forms are constructed as needed by removing suffixes to obtain a potential root and making sure that the potential root is in the dictionary and is marked to allow the removal of that suffix. If so, the inflected form is defined as having that root and features determined by the suffix in a regular way.

Having outlined the general structure of definitions, we can now look at the lexical categories in some detail. We distinguish two kinds of lexical categories, open and closed. Open categories are large, potentially infinite classes of words (such as nouns and verbs) which will never all be in the dictionary. These classes are quite productive, with new members arising almost daily, as technology progresses. The closed categories are finite, and, for the most part small, and they are not growing. These categories include prepositions, determiners, conjunctions, and modals, and we hope to cover these categories exhaustively at BBN. Thus, dictionary writers at DDC need only be concerned with the open categories, which we discuss more concretely below.

A. Open Category Properties.

N	=	noun (man, airplane, city)
NPR	=	proper noun (John Smith, USAF)
V	=	verb (walk, fly, see)
ADJ	=	adjective (tall, happy, green)
ADV	=	adverb (quickly, suddenly, certainly)

B. Closed Category Properties

CONJ	=	conjunction (and, or, but)
PREP	=	preposition (to, for, over)
PRO	=	pronoun (I, you, they)
DET	=	determiner (the, a, those)
ORD	=	ordinal (first, second, last, final)
NEG	=	negative (not)
COMP	=	comparative (more, less, greater)
OP	=	operation (plus, times)
QWORD	=	question noun (who, what, why)
QDET	=	question determiner (which, what)
MODAL	=	modal verb (should, would,)
INTEGER	=	integer (one, two, three)

III. Open Categories

A. N - Noun

The property N indicates that the word can be interpreted as a common noun. Every noun is mapped into its root form and supplied with an inflectional feature for number. This feature is encoded as follows:

(NU SG) if this is a singular form
(NU PL) if this is a plural form (e.g. OXEN)
(NU SG/PL) if this form is considered both singular and plural (e.g. FISH, SHEEP)

The property N should not have the value *; the feature (NU SG) is supplied by default for other legal atomic values.

The atomic arguments specify how the plural, if any, is formed The following atoms are recognized (the hyphens are required):

-S nouns with regular S plurals (e.g. BOOK, BOOKS, MULE, MULES)
-ES nouns with regular ES or IES (if the root ended in Y) plurals (e.g. CHURCH, CHURCHES, PONY, PONIES)
MASS mass and abstract nouns which have no plural (e.g. WATER, HEALTH)
IRR nouns whose plural form is irregular (e.g. DATUM, OX) (note that the plural form must have a separate dictionary entry)

The definitions given for the following words illustrate these conventions:

```
BOOK      (N -S)
HEALTH    (N MASS)
OX        (N IRR)
OXEN      (N (OX (NU PL)))
```

B. NPR - Proper Noun

Proper nouns have a very simple structure, since they do not have inflectional forms or features. The basic entry for a proper noun is (NPR *), although it is possible to use the root retrieving routines to provide a SUBSTITUTE effect. Thus if JOSEPH were defined as (NPR *), (NPR (JOSEPH)) and (SUBSTITUTE ((JOSEPH))) would be equivalent definitions for JOE.

C. V - Verb

The inflectional structure of verbs is more complicated than that of nouns. A verb is marked as a tensed form (present or past), an infinitive, and/or a participle (present or past). In addition, if the verb is marked as tensed, it must also be marked for person and number. These features are specified in the following way:

(TNS PRESENT) (TNS PAST	for tensed forms
(PRESPART T)	for present participles
(PASTPART T)	for past participles
(UNTENSED T)	for untensed infinitive forms
(PNCODE 3SG)	for third-person singular forms
(PNCODE XSG)	for every thing except the third-person singular
(PNCODE ANY)	for all person-number combinations

If the value of an inflectional feature is T, the T need not be specified. Thus, the following abbreviations may be used where appropriate: (PRESPART), (PASTPART), (UNTENSED). In addition, the grammar has been designed so that a tensed verb that has no PNCODE specified will be interpreted as if it has (PNCODE ANY) permitting the elimination of this very common feature.

As for nouns, the atom * is not a permissible property value. For other legal atomic values, the default features are (TNS PRESENT) (PNCODE X3SG) (UNTENSED), which correspond to

the normal behavior of the infinitive (root) form. The legal atomic values are (note the absence of an initial hyphen):

S-D — regular verbs which add S for the third-singular, D for the past tense and past participle, and ING for the present participle (e.g. INCLUDE, INCLUDES, INCLUDED, INCLUDED, INCLUDING).

S-ED — regular verbs like the above except that they add ED for the past tense and past participle (e.g. HAPPEN, HAPPENS, HAPPENED, HAPPENED, HAPPENING)

ES-ED — the same as S-ED except that the third-singular is formed with ES. Verbs that change Y to I and add ES or ED are also included. (e.g. PASS, PASSES, PASSED, PASSED, PASSING, STUDY, STUDIES, STUDIED, STUDIED, STUDYING)

IRR — infinitive forms of irregular verbs--all the other forms must have separate entries. (e.g. GIVE, MAKE, RUN)

Illustrative examples:

```
INCLUDE (V S-D)
HAPPEN  (V S-ED)
GIVE    (V IRR)
GAVE    (V (GIVE (TNS PAST)(PNCODE ANY)))
SAW  (V ( (SEE (TNS PAST))
          (SAW (TNS PRESENT) (PNCODE X3SG) (UNTENSED)) ))
SAWED (V (SAW (TNS PAST)))
```

Notice that if the root of one verb and an inflected form of another are homographs (i.e. they are spelled the same), the regular inflectional machinery cannot be used--all the forms of the homographic root must be explicitly defined. There is one other restriction: the features (UNTENSED) and (PASTPART) are mutually exclusive, so that the few verbs whose infinitive and past participle are the same must be handled specially, as illustrated below.

```
(RUN (V ( (RUN (TNS PRESENT) (PNCODE X3SG) (UNTENSED))
          (RUN (PASTPART)) ))
```

RUN is thus defined as an ambiguous verb whose two interpretations have the same root but different features.

D. ADJ - Adjective

Ordinary adjectives in English do not have any features, but many of them have inflected comparative and superlative forms. These are marked by the features (COMPARATIVE T) and (SUPERLATIVE T), which may be abbreviated (COMPARATIVE) and (SUPERLATIVE). Adjectives which do not admit these inflections in a regular way are simply marked as (ADJ *), for example, EXTREME and ESSENTIAL. Otherwise, adjectives can have the atomic values

R-ST if they form the comparative by adding R and the superlative by adding ST (e.g. CLOSE, CLOSER, CLOSEST)

ER-EST if they add ER and EST instead (e.g. PINK, PINKER, PINKEST) Adjectives which change Y to I are also included.

These atomic values do not supply any default features for the root form. Examples:

```
HAPPY    (ADJ ER-EST)
GOOD     (ADJ *)
BETTER   (ADJ (GOOD (COMPARATIVE)))
BEST     (ADJ (GOOD (SUPERLATIVE)))
```

It should be noted that nouns which can be used as modifiers need not be categorized as adjectives, since the grammar recognizes noun-noun modification.

E. ADV - Adverb

Like adjectives, adverbs can also be inflected for comparative and superlative, but the root itself has no features. Thus, irregular adverbs or adverbs that do not have comparatives or superlatives are marked (ADV *), while the regular forms use the same atomic value codes (ER-EST and R-ST) as adjectives. Examples:

```
HARD        (ADV ER-EST)
FAR         (ADV *)
FURTHER     (ADV (FAR (COMPARATIVE)))
```

IV. Syntactic Features

The property FEATURES is required in the definitions of root forms to specify the syntactic behavior of the root and all its inflected forms. At present, the grammar only examines FEATURES on verbs so that the property need not appear on roots in other categories. The value of FEATURES is a simple unordered list of atoms, each one denoting a different characteristic. The features which may be included for verbs are:

TRANS	if the verb can be transitive (e.g. HIT, KICK)
INTRANS	if the verb can be intransitive (e.g. WALK, GO)
INDOBJ	if the verb can take an indirect object (e.g. GIVE, BUY, TELL)
COPULA	if the verb can be a copular (i.e. can be followed by a predicate adjective) (e.g. BE, SEEM, APPEAR)
PASSIVE	if the verb can be passivized (e.g. DISCOVER, FIND) (Note: all PASSIVE verbs are TRANS, but not all TRANS verbs are PASSIVE--e.g. COST)

Appendix II contains a set of sentence frames which define these verbal characteristics.

With one exception, if a feature does not appear in the list, the grammar assumes that the verb does not have the characteristic in question. Thus, if the root WALK is not marked INTRANS, the grammar will not be able to parse the sentence "John walked." The exception is that in the special case when the only features a verb has are TRANS and PASSIVE, the whole FEATURES property may be ommitted. The following two definitions for KICK are equivalent:

```
KICK    (V S-ED FEATURES (PASSIVE TRANS))
KICK    (V S-ED)
```

Since a large proportion of verbs have only these two features, this convention should reduce the total size of the dictionary. Examples of complete dictionary entries:

```
GIVE      (V IRR FEATURES (PASSIVE TRANS INDOBJ))
GIVEN     (V (GIVE (PASTPART)))
GO        (V IRR FEATURES (INTRANS))
BECOME    (V IRR FEATURES COPULA))
```

V. SUBSTITUTE and COMPOUNDS

The properties SUBSTITUTE and COMPOUND change the words in the sentence, before the grammar has even looked at them. If none of the substitutions or compounds lead to a valid parse, the parser restores the sentence to its original form. In this case, the grammar examines the lexical category information in the word's definition; thus a definition can contain lexical category properties as well as SUBSTITUTE and COMPOUNDS.

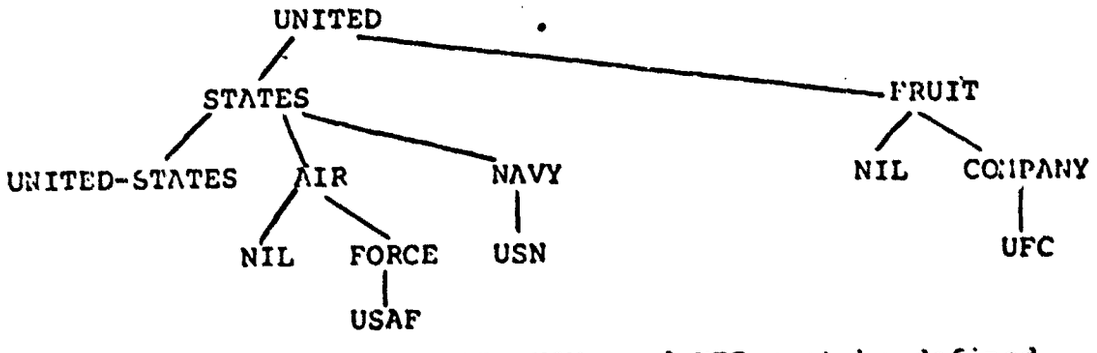
The value of SUBSTITUTE is a list of lists, each list being a possible string to be substituted for the word. Whereas COMPOUNDS causes a sequence of words to be replaced by a single word, SUBSTITUTE can have the opposite effect: If SRO were define as (SUBSTITUTE ((STANDING ROOM ONLY)), every occurrence of SRO in a sentence would effectively lengthen the string to be parsed.

As indicated earlier, COMPOUNDS provides a means of mapping idioms and compound expressions (sequences of words whose joint meaning is not simply the composition of the meanings of the individual words) into a single word representing the joint meaning. Thus in the earlier example, the sequence UNITED STATES was mapped into the "word" UNITED-STATES, which was then explicitly defined. The COMPOUNDS mechanism is general enough to handle arbitrarily long sequences and sequences which are identical to the initial

segments of longer sequences (e.g. UNITED STATES and UNITED STATES AIR FORCE). The various possibilities are expressed in the value of the COMPOUNDS property.

The compounds value (defined in Appendix I) can be thought of as a tree structure rooted in the word being defined (e.g. UNITED). Non-terminal nodes in the tree specify intermediate words in the compound expression, so that the non-terminal nodes encountered in tracing a path in the tree down from the root denote the sequence itself. The terminal node at the bottom of the path is the joint meaning of the sequence. There is a terminal node under each non-terminal to specify the joint meaning of subsequences that can occur independently; if the terminal is the atom NIL, then the non-terminal in question cannot be the last word in a sequence. The following example shows the value for COMPOUNDS and the corresponding tree structure necessary to recognize the expressions UNITED STATES, UNITED STATES AIR FORCE, UNITED STATES NAVY, and UNITED FRUIT COMPANY:

```
UNITED  (COMPOUNDS ((STATES UNITED-STATES (AIR NIL (FORCE USAF))
                                          (NAVY USN))
                    (FRUIT NIL (COMPANY UFC))))
```



Of course, UNITED-STATES, USAF, USN, and UFC must be defined separately (probably as (NPR *)).

62

VI. Formats for Dictionary Entries

The following is a formal specification of the syntax for dictionary definitions. The notation is similar to that used to describe context-free languages, except that nonterminal symbols are enclosed in angle-brackets and alternations are represented by the vertical bar. The only addition is the Kleene * operator, used to denote an arbitrarily (zero or more times) repeatable constituent.

(1) <definition> → (<definiens> <pair>*)

(2) <pair> → <lexical category> <legical category value> |
SUBSTITUTE <substitute value> |
FEATURES <feature value> |
COMPOUNDS <compounds value>

(3) <legical category> → N | V | ADJ | ADV ...

(4) <legical category value> → <morphology code> |
"*" |
<root-feature list> |
(<root-feature list> *)

(5) <root-feature list> → (<root> <inflectional feature> *)

(6) <inflectional feature> → (<inflectional feature name>
<inflectional-feature value>)

(7) <substitute value> → (<substitution> *)

(8) <substitution> → (<word> *)

(9) <feature value> → (<feature> *)

(10) <compounds value> → (<tree> *)

(11) <tree> → (<word> <result> <tree> *)

(12) <result> → <word> | NIL

<lexical category>, <morphology code>, <root>, <inflectional-feature name>, <inflectional-feature value>, <word>, and <feature> are all atoms as specified in the text. <definiens> is the word being defined.

VII. Frames for Syntactic Features

The following is a suggestive set of sentence frames for the determination of the syntactic features which must be specified in the definition of verbs. If a verb can fit into the open slot in a frame, then the root form of the verb must be marked with the syntactic feature (under the property FEATURES) with which the frame is associated. It should be noted that for some verbs it might be necessary to change the pronouns or substitute other noun-phrases in the frame in order to arrive at meaningful sentences; if a grammatical sentence results after these modifications, the verb must still be marked with the feature in question.

A. TRANS: A verb must be marked TRANS if it can be immediately followed by a direct object noun-phrase.
They _____ it. (e.g. "hit" but not "go")

B. INTRANS: A verb is intransitive if it does not require a direct object. (Note: a verb can be both TRANS and INTRANS, if the direct object is optional.)
They _____. (e.g. "ran", but not "surprised")

C. INDOBJ: A verb can take an indirect object and must be marked INDOBJ if (1) it can be followed by two noun-phrases, and (2) if interchanging the two noun-phrases and inserting the word "to" between them does not change the meaning of the sentence.
They _____ him it. (e.g. "gave", but not "hit")
They _____ it to him.

D. COPULA: A verb is a copula if it can be immediately followed by an adjective which is predicated of the subject.

They _____ tall. (e.g. "are", but not "weigh")

E. PASSIVE: Transitive verbs which can be passivized must be marked both TRANS and PASSIVE.

He _____ them.
They were _____ by him. (e.g. "see (saw, seen) but not "cost")

Appendix F.

The DDC Trial Sample

```
(PRIN1 (QUOTE "CREATED ") T)
(PRIN1 (QUOTE "3 -JUN-71 19:04:09") T)
(TERPRI T)
  (PRINT (QUOTE ((VARS TRIALSENTENCES)))
         T)
  (RPAQQ TRIALSENTENCES
         (

          (* UNPRIMED SENTENCES CORRESPOND TO THOSE ENTERED BY
          DDC; SINGLE PRIMED SENTENCES CORRESPOND TO THE ABOVE
          WITH CORRECTED PUNCTUATION, I.E. ACCORDING TO THE
          CONVENTIONS OF THIS SYSTEM;
          DOUBLE PRIMED SENTENCES CORRESPOND WORD FOR WORD
          WITH THE USER'S ACTUAL REQUEST; TRIPLY PRIMED
          SENTENCES CORRESPOND TO THE REQUESTS MINUS PADDING)


          (48111 THE EFFECTS OF CORROSION FATIGUE ON MILITARY VEHICLES
                 WITH PARTICULAR INTEREST IN THE EXTENT OF THIS TYPE OF
                 MECHANICAL FAILURE IN THE FIELD)
          (48125 INERTIAL GUIDANCE SYSTEMS AND SENSORS DEVELOPMENT AND
                 TEST PRIMARILY CONVENTIONAL INERTIAL PLATFORMS,
                 GIMBALLESS INERTIAL PLATFORMS AND DATA PROCESSING
                 SCHEMES, SUCH AS (* KALMAN-SCHMIDT FILTERING))
          (48137 BEST METHODS TO GROW AND PROPERTIES OF ALKALI IODATES:
                 LITHIUM IODATE, SODIUM IODATE)
          (48139 REACTIONS OF POLYPERFLUORINATED: COMPOUNDS, MATERIALS,
                 POLYMERS, AND THERMALLY STABLE FLUIDS)
          (48139' REACTIONS OF POLYPERFLUORINATED COMPOUNDS, MATERIALS,
                  POLYMERS, AND THERMALLY STABLE FLUIDS)
          (48197 DESCRIPTIONS OF WORK DONE ON LIQUID ROCKET ENGINES
                 WHICH THROTTLE OVER A (* 10:1)
                 RANGE WHICH UTILIZE
                 (* LO2/H2)
                 PROPELLANTS)
          (48256 DYNAMIC SIMULATION OF ROCKET ENGINES, STEADY-STATE
                 SIMULATION OF ROCKET ENGINES, COMPUTER MODEL OF ROCKET
                 ENGINES)
          (48256 METHODS OF ELECTRONICALLY SCANNING, STEERING, AND/OR
                 TRAINING, TRANSMITTING AND RECEIVING TRANSDUCER ARRAYS)
          (48257 PHASED ARRAYS, LENS, (* ATCRBS)
                 , AIR TRAFFIC CONTROL RADAR BEACON SYSTEM, AZIMUTH
                 SCANNING, ELECTRONIC SCAN, AGILE BEAM)
          (48259 HOW DO GEOPOTENTIAL FORCES AFFECT SYNCHRONOUS
                 SATELLITES)
          (48263 DYNAMIC SIMULATION DESCRIBING POWER TRANSMISSION
                 THROUGH A PLANETARY DIFFERENTIAL GEAR TRAIN, DRIVING
                 LOADS WHICH INVOLVE INERTIA AND FRICTION)
          (48267 OPERATIONAL CHARACTERISTICS AND SHIPBOARD INTERFACE OF
                 NAVAL WEAPONS AND FIRE CONTROL SYSTEMS INCLUDING THE
                 WEAPONS, GUNS, MISSILES, DEPTH CHARGES, TORPEDOES)
```

(48267' OPERATIONAL CHARACTERISTICS AND SHIPBOARD INTERFACE
OF NAVAL WEAPONS AND FIRE CONTROL SYSTEMS, INCLUDING
THE WEAPONS, GUNS, MISSILES, DEPTH CHARGES, TORPEDOES)
(48267'' OPERATIONAL CHARACTERISTICS AND SHIPBOARD INTERFACE
OF NAVAL WEAPONS AND FIRE CONTROL SYSTEMS
(GUNS, MISSILES, DEPTH CHARGES, TORPEDOS, ETC.))
(48268 DYNAMIC STRAIN MEASUREMENT AND HOW IT IS AFFECTED BY
STRAIN GAGE GRID LENGTH, BONDING ADHESIVES, AND STRAIN
GAGE BACKING)
(48271 ELECTRONIC SWITCHING, (↑ TDM)
SWITCHING,
(↑ PCM)
SWITCHING, DIGITAL TRANSMISSION,
(↑ T-1)
LINE)
(48283 PLEASE PROVIDE INFORMATION ON THE MOST CURRENT
SCIENTIFIC RESEARCH BEING DONE IN THE AREA OF TORNADOS)
(48285 CARGO SHIP AUTOMATIC STABILIZATION SYSTEM)
(48293 TECHNICAL OBSOLESCENCE, MANAGEMENT OBSOLESCENCE)
(48294 STORAGE AND RETRIEVAL DATA SYSTEMS, DATA BASE
MANAGEMENT)
(48294' STORAGE AND RETRIEVAL DATA SYSTEMS; DATA BASE
MANAGEMENT)
(48295 PYROPHORIC MATERIALS : GENERAL PROPERTIES (CHEMICAL
AND
PHYSICAL)
, SAFETY MEASURES REQUIRED FOR PACKAGING AND STORAGE,
AND THE USE IN MILITARY PYROTECHNIC SYSTEMS)
(48296 LISTENING AND LEADERSHIP, DEVELOPING LISTENING POWER,
EFFECTIVE LISTENING, HEARING VERSUS LISTENING)
(48298 FIRE BOMBS WEIGHING 2000 POUNDS OR MORE)
(48300 ARMY SURFACE TO AIR MISSILE SYSTEM)
(48301 REACTIVITY STUDIES INVOLVING ALUMINUM AND IRON, THEN
STEEL AND ALUMINUM ALLOYS, IN THE PRESENCE OF HOT
GASES)
(48302 CARRIER RECOMBINATION AS PRODUCED BY GOLD IN GAAS)
(48307 HEART RATE)
(48306 GALVANIC SKIN RESPONSE)
(48309 BLOOD PRESSURE)
(48311 TRAVELING SALESMAN PROBLEM, SEQUENCING, DECISION
THEORY, DYNAMIC PROGRAMMING, GAME THEORY)
(48315 ZIRCONIUM USE IN PYROTECHNICS
(AS IN CHEMICAL RELEASES AND FLASH CHARGES))
(48323 DRYING OF AIR IN FREEZE DRYING OF FOODS)
(48324 LOW COST DESIGN CONCEPTS AND LOW COST MANUFACTURING
METHODS FOR SMALL MISSILE SYSTEMS SUCH AS (↑ SRAM)
, SHORT RANGE ATTACK MISSILE,
(↑ HOUNDOG))
(48352 RAMJET COMBUSTION WITH EMPHASIS ON FLAME HOLDERS OF
THE BLUNT BODY AND CATALYTIC TYPES)

(48374 THE ELECTRICAL AND MAGNETIC CONDUCTIVITY OF BACTERIA AND FUNGI)
(48382 ANY DOCUMENTS DEALING WITH ANY TYPE OF COMPUTER LANGUAGES)
(48389 EFFECTS OF TEMPERING ON 4340 STEEL)
(48424 VAPORIZATION, MASS SPECTROGRAPHIC DATA ON ALUMINUM OXIDES, SILICON OXIDES, AND SILICON ALUMINUM OXIDES)
(48485 ALLOYING OR COMPOSITING SODIUM METAL WITH OTHER METALS AND MATERIALS)
(48741 METHODS OF PROVIDING ADEQUATE AND COMPLETE DEFENSE OF ISOLATED SIGNAL UNITS AND INSTALLATIONS LOCATED ON MOUNTAIN TOPS AND WITHIN AN INTERNAL DEFENSE ENVIRONMENT)
(59813 SHIPS DESIGN RELATED TO NAVY SHIPBUILDING PROGRAM)
(59815 NICKEL ALLOY (↑ NI34TI)
, NICKEL ALLOY
(↑ NITINOL)
, OR NICKEL ALLOY
(↑ NI34TI))
(59816 TRACE ELEMENTS IN LAKES, RIVERS AND STREAMS)
(59817 CONCENTRATION OF HEAVY METALS (RESULTS, NOT ANALYTICAL METHODS)
IN LAKES, STREAMS, RIVERS, OCEANS)
(59818 ELECTROLYTIC AMMINATION OF AROMATIC COMPOUNDS)
(59819 HANDLING, SAFETY AND PRESERVATION OF LIQUID (↑ N2))
(59823 THE EFFECTS OF NOISE ON HUMANS, INCLUDING PSYCHOLOGICAL, PHYSIOLOGICAL, ANNOYANCE)
(59825 ALL INFORMATION ON THE SUBJECTS OF NATURAL GAS, TRANSMISSION PIPELINE, FLOW DYNAMICS, PRESSURE TRANSIENTS AND SIMULATIONS OF THE ABOVE SUBJECTS)
(59862 LITERATURE SEARCH ON HYDRODYNAMIC STABILITY AND CONTROL DERIVATIVES)
(59867 SPACE COMMUNICATIONS, SPACECRAFT COMMUNICATIONS, LASER COMMUNICATIONS, MICROWAVE COMMUNICATIONS, OPTICAL COMMUNICATIONS, PULSE COMMUNICATIONS, RADIO COMMUNICATIONS, SECURE COMMUNICATIONS, RADIO TELEMETRY, COMMUNICATION SATELLITE)
(59868 INFORMATION RELATIVE TO ELECTROSTATIC GYROSCOPE DESIGN, ANALYSIS, ERROR MODELS, AND TESTING)
(59869 (↑ WALSH)
FUNCTIONS AND THEIR APPLICATIONS, SPECIFICALLY COMPUTER PROGRAMS WHICH CONTAIN FAST FOURIER TRANSFORMS USING THE
(↑ WALSH)
FUNCTIONS)
(59872 SEA MINES, MINE WARFARE, MINE SWEEPER)
(59875 LIQUID CRYSTALS)
(59876 ENVISCID SPUN METAL, METALLIC FIBERS, METAL FILAMENTS, MISSILE WIRE AND MISSILE TOW CABLE, MELT-SPUN METAL FIBERS, SPUN METAL, STEEL FIBER FOR USE IN TIRE CORD, FINE-DIAMETER CABLE, OR FINE-DIAMETER WIRE)

(59877 VOLUME, WEIGHT, (↑ RPM)
, POWER, DESIGN INFORMATION ON SINGLE REDUCTION GEAR SYSTEMS)
(59878 AIRCRAFT TYPE: F-15)
(59883 DISPLAYS AND CONTROLS FOR WEAPONS DELIVERY FROM AIRCRAFT)
(59942 THEORY AND APPLICATIONS OF (↑ WALSH)
FUNCTIONS AND
(↑ HADAMARD)
MATRICES)
(59948 REPORT BIBLIOGRAPHY PERTAINING TO BOUNDARY LAYER CONTROL AND JET FLAPS AT HIGH SUBSONIC AND SUPERSONIC SPEEDS IS REQUIRED)
(59948''' BIBLIOGRAPHY PERTAINING TO BOUNDARY LAYER CONTROL AND JET FLAPS AT HIGH SUBSONIC AND SUPERSONIC SPEEDS)
(59951 CURRENT RESEARCH ON ANAEROBIC FILTERS FOR SEWAGE TREATMENT PLANTS)
(59953 POPULATION GENETICS, EVOLUTION GENETICS, OR MATHEMATICS GENETICS)
(59955 'NCHOIC COOLING FOR SHIPS)
(59956 TYPES OF EXPLOSIVES USED FOR EXPLOSIVE CLADDING, FORMING AND WELDING INCLUDING THERMOPHYSICAL PROPERTIES AS BURNING RATE)
(59956' TYPES OF EXPLOSIVES USED FOR EXPLOSIVE CLADDING, FORMING AND WELDING, INCLUDING THERMOPHYSICAL PROPERTIES SUCH AS BURNING RATE)
(59957 AIR FORCE REPORTS ON THE HH-53C HELICOPTER: DROP TANKS, RESCUE HOIST, APPROACH COUPLER, DOPPLER NAVIGATION)
(59958 OPERATIONS ANALYSIS: ALLOCATION MODELS, (↑ CRITICAL PATH METHOD)
, ECONOMIC MODELS, NETWORK FLOWS, OR SYSTEMS ANALYSIS)
(59963 INLET ICING; AIR INLETS, COWLING, AIR SCOOPS, AUXILIARY AIR INLETS, OR ACCESSORY INLETS)
(59964 DESIRE INFORMATION ON METHODS OR TECHNIQUES FOR HIGH DENSITY MAGNETIC TAPE RECORDING)
(59966 BIBLIOGRAPHY ON RISK ANALYSIS)
(59967 DESIGN AND PERFORMANCE DATA FOR BUTTERFLY VALVES; ROUND, SQUARE, ELLIPTICAL, IRREGULAR SHAPES)
(59967' DESIGN AND PERFORMANCE DATA FOR BUTTERFLY VALVES: ROUND , SQUARE , ELLIPTICAL , OR IRREGULAR SHAPES)
(59973 METHOD OF MEASUREMENT OF BURST HEIGHT OF PROJECTILES, 40MM UP TO 16 INCHES)
(59974 POLAR ACTIVITY)
(59979 AIRCRAFT GUN MUZZLE BLAST EFFECTS AND CONTROL)
(59987 INFORMATION REQUIRED ON THE REPAIR AND SUPPLY OF MICROMINIATURE ELECTRONIC COMPONENTS, SUBASSEMBLIES AND ASSEMBLIES)
(59981 BUOY MOTIONS; CABLE DYNAMICS, STRESS ANALYSIS OF CABLE, AND CABLE MOTIONS)

(59981' BUOY MOTIONS: CABLE DYNAMICS, STRESS ANALYSIS OF CABLE, AND CABLE MOTIONS)
(59952 PHASED ARRAY RADARS)
(59985 MICRO-YIELD AND MICRO-CREEP OF MATERIALS)
(59986 BIBLIOGRAPHY ON UNDERGROUND NUCLEAR POWER PLANTS AT MAJOR POWER LEVELS BOTH IN THE UNITED STATES AND EUROPE PARTICULARLY SWEDEN)
(59986' BIBLIOGRAPHY ON UNDERGROUND NUCLEAR POWER PLANTS AT MAJOR POWER LEVELS BOTH IN THE UNITED STATES AND IN EUROPE, PARTICULARLY SWEDEN)
(59989 DRILLING FUNDAMENTALS, DRILL LIFE, HOLE ACCURACY AND STRAIGHTNESS OF HOLE; ALSO, VIBRATIONS IN DRILLING MACHINES)
(6[illegible]072 BIOLOGICAL EFFECTS OF LIGHT AND LASERS)
(6[illegible]071 BACKGROUND INFORMATION ON THE TARGET DETECTING DEVICE FOR THE TALOS MISSILE)
(60073 THE EFFECTS OF ELECTROLYTIC MACHINING ON THE METAL SURFACE INTEGRITY OF IRON, NICKEL AND COBALT BASE ALLOYS)
(60075 LASER DAMAGE OF GLASS AND/OR THIN FILM OPTICAL COATINGS)
(60077 DYNAMIC PROGRAMMING; ALLOCATION MODELS, CONSTRAINTS, STEEPEST DESCENT METHOD)
(60077' DYNAMIC PROGRAMMING: ALLOCATION MODELS, CONSTRAINTS, STEEPEST DESCENT METHOD)
(60078 B-52-H RESPONSE TO ATMOSPHERIC TURBULENCE)
(60011 ULTRASONIC EQUIPMENT)
(60017 ISOSTATIC AND HYDROSTATIC PRESSING OF SECONDARY HIGH EXPLOSIVES WITH VARIOUS DWELL AND RECYCLE TIMES)
(60020 EFFECTS OF CNS DRUGS ON REPEATED TESTING PROCEDURES IN THE SAME HUMAN SUBJECT)
(60021 OPTICAL ELECTRONIC MEMORY ELEMENTS)
(60022 HOLOGRAPHY AS A COMPUTER MEMORY DEVICE)
(60046 CARGO COMPARTMENT TEMPERATURES FOR C-130 AIRCRAFT)
(60047 OCEAN AND SHALLOW WATER CURRENT METERS)
(60051 MANAGEMENT MONITORING OF QUALITY CONTROL)
(60053 SHOCK WAVE PROFILE SHAPES IN SUPERSONIC AND HYPERSONIC FLOW)
(60054 METHODS USED TO IDENTIFY FORMAL SCHOOLS INSTRUCTORS FOR INDENTIFICATION AFTER COMPLETION OF A TOUR OF DUTY)
(60057 .45 PISTOL AND .38 CAL. SPECIAL PISTOL)
(60058 USE OF EXPLOSIVES TO JOIN METALS, INCLUDING TECHNIQUES AND TYPES OF EXPLOSIVE MATERIALS)
(60067 BIBLIOGRAPHY ON SOVIET SUBMARINE CHARACTERISTICS: RADIATED NOISE, AND SPECTRUM ANALYSIS)
(60076 BIBLIOGRAPHY ON LAVA SYSTEM)
(60082 UNIVERSAL AIRCRAFT FLIGHT SIMULATORS, TRAINERS, OR FLIGHT RESEARCH TOOLS)
(60085 ALL DOCUMENTS PERTAINING TO SPEECH INTELLIGIBILITY, VOICE COMMUNICATIONS JAMMING, AND SPEECH INTERFERENCE)
(60088 COMPUTER PROGRAMMING PROCEDURES AND STANDARDS)
(60090 HIGH TEMPERATURE RESINS AND POLYMERS, AND GRAPHITE AND BORON COMPOSITE MATERIALS)))

STOP